NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NETWORKED HUMANOID ANIMATION DRIVEN BY HUMAN VOICE USING EXTENSIBLE 3D (X3D), H-ANIM AND JAVA SPEECH OPEN STANDARDS

by

Ozan Apaydın

March 2002

Thesis Advisor:	Don Brutzman
Second Reader:	Xiaoping Yun

This thesis done in cooperation with the MOVES Institute
Approved for public release; distribution is unlimited

Report Documentation Page

Report Date 29 Mar 2002	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Networked Humanoid Animation Driven by Human Voice Using Extensible 3D (X3D), H-ANIM and JAVA Speech Open Standards	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Apaydin, Ozan	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Naval Postgraduate School Monterey, California	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor’s Acronym(s)**
	Sponsor/Monitor’s Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes The original document contains color images.	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 102	

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Networked Humanoid Animation Driven by Human Voice Using Extensible 3D (X3D), H-Anim and Java Speech Open Standards	**5. FUNDING NUMBERS**
6. AUTHOR Apaydın, Ozan	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT

Speech-recognition technology is beginning to be used in automobiles, telephones, personal digital assistants (PDAs), medical records, e-commerce, text dictation and editing. Speech recognition can also be integrated into Virtual Environments (VEs) to create responsive virtual entities. Like the mouse, keyboard, and the trackball, Speech-recognition technology can enhance the control of a computer and improve communication.

Dramatically expanding interest in the Internet and VEs has been gated by limited interactivity with human-avatar models. As more users begin interacting with avatars in VEs, designers are prompted to create more realistic, humanlike avatars. This quest for realism needs to go beyond visual aspects to include speech-recognition technology, which can greatly augment the realism of these avatars.

This thesis presents design and development of a Voice User Interface (VUI), which maps to a set of behavioral motions for humanoid avatars using Extensible 3D (X3D) graphics, the Virtual Reality Modeling Language (VRML), Humanoid Animation (H-Anim) Standard and Java Speech API. The VUI includes a suitable speech-recognition component for application-command vocabularies. This thesis also demonstrates interchangeability of both avatars and animation behaviors, and creates networked humanoid animation driven by a human voice.

14. SUBJECT TERMS Humanoid Animation (H-Anim) Specification, Avatars, X3D, X3D-Edit, VRML, Java, Java Speech API, Java Speech Grammar Format, Web3D Consortium, Voice User Interface (VUI).	**15. NUMBER OF PAGES** 102
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

NETWORKED HUMANOID ANIMATION DRIVEN BY HUMAN VOICE USING EXTENSIBLE 3D (X3D), H-ANIM AND JAVA SPEECH OPEN STANDARDS

Ozan Apaydın
Lieutenant Junior Grade, Turkish Navy
Undergraduate (B. S.), Turkish Naval Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2002

Author:
Ozan Apaydın

Approved by:
Don Brutzman
Thesis Advisor

Xiaoping Yun
Second Reader

Chris Eagle
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Speech-recognition technology is beginning to be used in automobiles, telephones, personal digital assistants (PDAs), medical records, e-commerce, text dictation and editing. Speech recognition can also be integrated into Virtual Environments (VEs) to create responsive virtual entities. Like the mouse, keyboard, and the trackball, Speech-recognition technology can enhance the control of a computer and improve communication.

Dramatically expanding interest in the Internet and VEs has been gated by limited interactivity with human-avatar models. As more users begin interacting with avatars in VEs, designers are prompted to create more realistic, humanlike avatars. This quest for realism needs to go beyond visual aspects to include speech-recognition technology, which can greatly augment the realism of these avatars.

This thesis presents design and development of a Voice User Interface (VUI), which maps to a set of behavioral motions for humanoid avatars using Extensible 3D (X3D) graphics, the Virtual Reality Modeling Language (VRML), Humanoid Animation (H-Anim) Standard and Java Speech API. The VUI includes a suitable speech-recognition component for application-command vocabularies. This thesis also demonstrates interchangeability of both avatars and animation behaviors, and creates networked humanoid animation driven by a human voice.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr. Don Brutzman, and Dr. Xiaoping Yun, for their guidance, support and motivation throughout this study. I would like to thank Cindy Ballreich, Allen Dutton and James Smith for allowing the use of their human models.

I would like to thank to my wife (Filiz APAYDIN) for her unending patience and full support. Without the loving support of my wife, this work could not have been completed.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The arrow which was shot,
The word which was spoken and
The life which passed never return.
- A Turkish Proverb

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.
"The Road Not Taken" by Robert Frost (1920)

A. VOICE-INTERFACES BACKGROUND

In 1986, Dr. Jacob Nielsen asked a group of 57 IT professionals to predict what would be the greatest changes in user interfaces by the year 2000. The top-five answers were

Speech I/O	33%
Individualized interaction	19%
Increased use of graphics, mice, icons, etc.	16%
Dialogues developed by the users themselves	12%
Other new I/O-media than speech	12%

Table 1.1 User Interfaces Prediction Table [From Nielsen 99]

While Graphical User Interfaces (GUIs) have clearly been the winner since that time, Voice User Interfaces (VUIs) certainly failed to reach the demand that IT professionals expected. The key issue in interaction design and the main determinant of usability is what the user "says" to the interface. Whether he or she provides the command by speaking or by typing matters little to the user. Thus, having voice interfaces will not necessarily free us from the most substantial part of user interface design: determining the structure of the dialogue, what commands or features are available, how the users are to specify what they want, and how the computer is to communicate the feedback.

Voice interfaces have their greatest potential in the following cases:

- Users with various physical disabilities that prevent them from using a mouse or keyboard. Such users are not able to drive an avatar in a Virtual World.

- All users, with or without disabilities, whose hands and eyes are occupied with other tasks. For example, while driving a car or while repairing a complex piece of equipment.
- Users who do not have access to a keyboard and/or a monitor. For instance, users accessing a system through a payphone.

B. MOTIVATION

This thesis provides a step in a different direction to support virtual worlds by offering another user-control option other than mouse, keyboard, and trackball. Five examples of possible uses of VUIs are examined: SALT Reports, entertainment, virtual environments, e-commerce and gun control. Specific details regarding these uses follow:

1. Exemplar: Situation Reports Using Voice Recognition

The U.S. Army is exploring the use of speech-recognition software, which is seen increasingly in the commercial sector, as a potentially useful tool in future combat systems. The Army plans to integrate tactical voice activation into Version Four of Force XXI Battle Command, Brigade and Below software (FBCB2). FBCB2 is a digital command and control system, mounted on tactical vehicles and other platforms to provide battle command and situational awareness information to the war fighter, from brigade level down to the soldier platform level. FBCB2 is the central system of the future Army Battle Command Systems, a collection of systems that provides the capability to the war fighter collectively known as "network-centric warfare." [Seffers 2001]

The operational requirements for tactical voice activation in FBCB2 call for a voice-activated means to perform key operations. This would include situation and SALT (Size, Activity, Location, Time) reports, in which soldiers report the size, activity, location and time of enemy forces spotted on the battlefield. For example, the soldier spotting an enemy armored personnel carrier might tell the computer: “SALT report.” The computer prepares to receive information, and the soldier might say “Enemy equipment type: APC. Quantity: Six. Enemy Activity: Attacking.” [Seffers 2001]. The computer converts such information into a format for transmission via pre-established or directed distribution list that might include the unit commander, intelligence officers and artillery forces.

2. Entertainment

VUIs may be integrated into video games to control the present entities. Imagine a tactical war game in which the troops can be directed by saying, “Go to the end of the street!” or “Open fire!” Imagine a game that you could give tactical orders to the commanders of your military forces with your own voice. Using natural language processing (NLP), the virtual commanders might understand your commands and carry out the missions. Examples in this area can be expanded easily.

3. Virtual Environments (VEs)

VUIs are an appealing alternative while interacting with Virtual Environments (VEs). For example, intelligent avatars might replace actual bridge personnel in a warship bridge simulation. These avatars might understand voice commands, fulfill them and report via their own speech-recognition capabilities. Such an approach contributes to the simulation in two ways. The first is that the simulation environment becomes more realistic. The second is that the simulation becomes trainee independent. In other words, no bridge personnel are needed to run the simulation. Additionally, inserting intelligent avatars to the VEs can provide a range of possibility from non-person-collaborative simulation to many-person-collaborative simulation.

4. E-Commerce

E-Commerce is growing rapidly on the Internet, and speech-recognition technology can be integrated into advertising. An avatar, which can speak by synthesizing text from a database into speech, can present the products of an e-commerce company. This can engage e-customers both visually and aurally.

5. Gun Control

VUIs can also be used in gun-control systems. For example, if the fire controls of a warship or a tank gun have a speech-recognition capability, the controller can tell the gun to slew right and fire. Then a diagnosis on the gun's condition may be asked. Obviously, recognition accuracy becomes a critical requirement.

C. THESIS GOALS

The overall goals of this thesis are to

- Perform a background search on speech-recognition technology to find a suitable component for this project,
- Develop a VUI (Voice User Interface) that maps between human voice commands and a set of animations of the avatar, thus providing voice access to an animation application,
- Build a motion library to animate available humanoids,
- Demonstrate interchangeability of the behaviors and the humanoids,
- Create an integrated humanoid animation application driven by a human voice.

D. THESIS ORGANIZATION

Six chapters comprise this research:

- *Chapter I–Introduction*: Identifies the purpose and motivation behind conducting this research. Establishes the goals for the thesis.
- *Chapter II–Related Work*: Provides information on humanoid models, and the previous research conducted in this area.
- *Chapter III–Speech-Recognition Technology*: Introduces background information and basic concepts of Speech Recognition.

- *Chapter IV–Java Speech API*: Provides an overview for the Java Speech API and describes the speech technologies that are supported through Java Speech API.

- *Chapter V–Implementation: Building Voice-Enabled Humanoid Animation*: Describes the general system structure, software components and implementation process.

- *Chapter VI–Conclusion and Recommendations Future Work*: Explains the conclusions and provides recommendations regarding possible future work.

II. RELATED WORK

A. INTRODUCTION

This chapter provides an overview to Virtual Reality Modeling Language (VRML), Extensible 3D (X3D) and Kinematics. It further examines Humanoid Animation Working Group, Humanoid Animation Specifications and Humanoid Models.

B. VIRTUAL REALITY MODELING LANGUAGE (VRML)

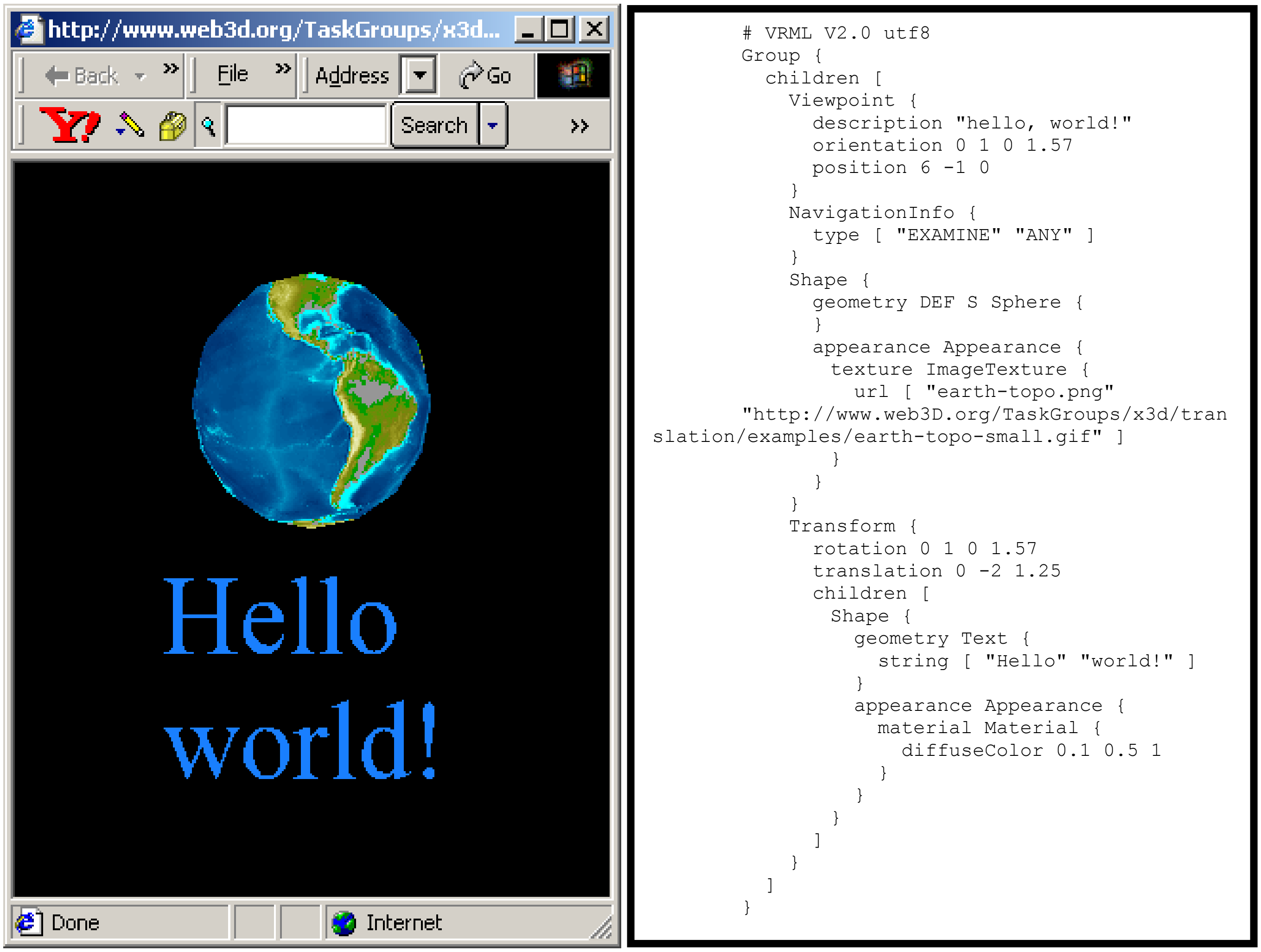
The Virtual Reality Modeling Language (VRML) is a file format for describing interactive 3D objects and worlds. VRML is designed to be used on the Internet, intranets, and local client systems. VRML is also intended to be a universal interchange format for integrated 3D graphics and multimedia. Moreover VRML may be used in a variety of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages and shared virtual worlds. VRML is capable of representing static and animated dynamic 3D and multimedia objects with hyperlinks to other media such as text, sounds movies and images. The VRML Specification is an International Standards Organization (ISO) specification (ISO/IEC 14772-1:1997). VRML also provides a large number of 3D graphics nodes, which are organized in a hierarchy within a file, to compose a directed acyclic graph (DAG) called a scene graph [Brutzman, 1998].

VRML file extension is `.wrl` or `.wrz` (if the file is gzip-compressed). A VRML file can contain four main types of components:

- The VRML Header,
- Prototypes,
- Shapes (geometry and appearance), Interpolators, Sensors and Scripts,
- ROUTEs. [Ames, 1997]

The only required element that VRML file must contain is the VRML header. Prototypes (PROTOs) allow a user to author new 3D graphic node types and can be formed into libraries and reused by referencing them inside an external prototype

(EXTERNPROTO). Shapes encompass both object geometry and appearance. Sensors allow users to interact with the scene. Script nodes provide an interface between the VRML scene and a program script; usually written in Java or JavaScript (i.e. ECMAScript). Script nodes are very important in creating complex actions and animations. Finally, routes are statements, which define connections between named nodes and fields by allowing events to be passed from source to target.



```
        # VRML V2.0 utf8
        Group {
          children [
            Viewpoint {
              description "hello, world!"
              orientation 0 1 0 1.57
              position 6 -1 0
            }
            NavigationInfo {
              type [ "EXAMINE" "ANY" ]
            }
            Shape {
              geometry DEF S Sphere {
              }
              appearance Appearance {
                texture ImageTexture {
                  url [ "earth-topo.png"
        "http://www.web3D.org/TaskGroups/x3d/tran
slation/examples/earth-topo-small.gif" ]
                }
              }
            }
            Transform {
              rotation 0 1 0 1.57
              translation 0 -2 1.25
              children [
                Shape {
                  geometry Text {
                    string [ "Hello" "world!" ]
                  }
                  appearance Appearance {
                    material Material {
                      diffuseColor 0.1 0.5 1
                    }
                  }
                }
              ]
            }
          ]
        }
```

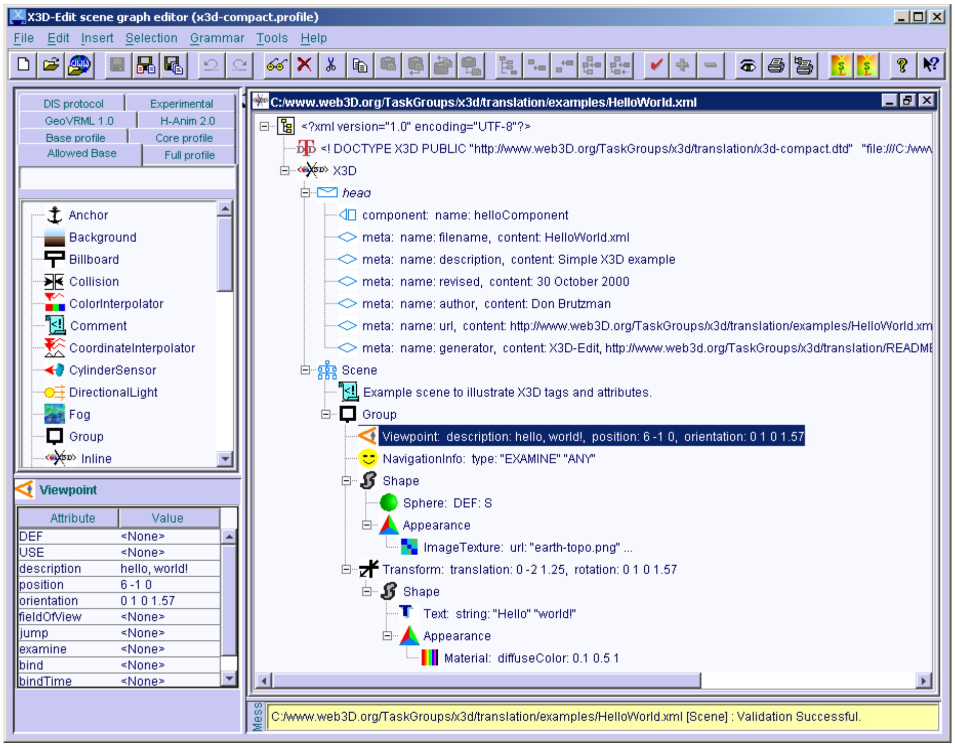
Figure 2.1 HelloWorld.wrl Scene and Source [Brutzman, 2000]
http://www.web3d.org/TaskGroups/x3d/translation/examples/HelloWorld.x3d, .wrl

C. EXTENSIBLE 3D (X3D) GRAPHICS SPECIFICATION AND X3D-EDIT AUTHORING TOOL

The X3D Graphics Working Group is designing and implementing the next-generation Extensible 3D (X3D) Graphics specification. The X3D is scene graph architecture and encoding that improves on the Virtual Reality Modeling Language (VRML) international standard (VRML 97, ISO/IEC 14772-1:1997). X3D uses the Extensible Markup Language (XML) to express the geometry and behavior capabilities of VRML. X3D is thus a backward-compatible XML tagset for describing the VRML 200x standard for Web-capable 3D content. Such content is not static but dynamic, driven by a rich set of interpolators, sensor nodes, scripts, and behaviors. [Brutzman, Blais, Horner, Nicklaus 2001]

X3D-Edit is a graphics file editor for Extensible 3D (X3D) that enables simple error-free editing, authoring and validation of X3D or VRML scene-graph files. Context-sensitive tooltips provide concise summaries of each VRML node and attribute. These tooltips simplify authoring and improve understanding for both novice and expert users. X3D-Edit is being used to develop and test the Extensible 3D (X3D) tagset for the next-generation Virtual Reality Modeling Language (VRML 200x).

X3D-Edit uses the XML tagset defined by the X3D Compact Document Type Definition (DTD) in combination with Sun's Java, IBM's Xeena XML editor, and an editor profile configuration file.



```
<?xml version="1.0" encoding="UTF-8"?>
  <! DOCTYPE X3D PUBLIC "http://www.web3D.org/TaskGroups/x3d/translation/x3d-compact.dtd"  "file:///C:/www
  X3D
    head
      component:  name: helloComponent
      meta:  name: filename,  content: HelloWorld.xml
      meta:  name: description,  content: Simple X3D example
      meta:  name: revised,  content: 30 October 2000
      meta:  name: author,  content: Don Brutzman
      meta:  name: url,  content: http://www.web3D.org/TaskGroups/x3d/translation/examples/HelloWorld.xm
      meta:  name: generator,  content: X3D-Edit, http://www.web3d.org/TaskGroups/x3d/translation/READMI
    Scene
      Example scene to illustrate X3D tags and attributes.
      Group
        Viewpoint:  description: hello, world!,  position: 6 -1 0,  orientation: 0 1 0 1.57
        NavigationInfo:  type: "EXAMINE" "ANY"
        Shape
          Sphere:  DEF: S
          Appearance
            ImageTexture:  url: "earth-topo.png" ...
        Transform:  translation: 0 -2 1.25,  rotation: 0 1 0 1.57
          Shape
            Text:  string: "Hello" "world!"
            Appearance
              Material:  diffuseColor: 0.1 0.5 1
```

Attribute	Value
DEF	<None>
USE	<None>
description	hello, world!
position	6 -1 0
orientation	0 1 0 1.57
fieldOfView	<None>
jump	<None>
examine	<None>
bind	<None>
bindTime	<None>

C:/www.web3D.org/TaskGroups/x3d/translation/examples/HelloWorld.xml [Scene] : Validation Successful.

Figure 2.2 Screen Shot of X3D-Edit While Editing HelloWorld.xml

D. KINEMATICS

Kinematics is the study of motion, in particular of the relationships of the various quantities of motion to one another. Specifically, kinematics deals strictly with position and orientation (i.e. posture). In contrast, dynamics considers forces and moments in combination with kinematics. At first glance, kinematics may seem like 17th century physics, but in fact it is the formal basis of almost all translation/scaling/rotating motion. Kinematics can provide a model of the human skeleton as well as of a steam engine. As such, kinematics is a fundamental science for the computer animator because it studies and catalogs knowledge about motion–the continuous change of place or position. [Pocock, Rosebush 2002]

The most basic kinematic element for articulated or rigid body kinematics is the *link,* which is a rigid moving part. Links are joined into *linkages* by the use of joints. The simplest kind of linkage is the *kinematic pair*, which consists of two links organized so that one is constrained to rotate about a pivoting joint, or to move back and forth inside a slide. The kinematic pair is the nucleus of kinematics. Connecting kinematic pairs with joints produces a *kinematic chain*, also called *articulated chain* or simply *chain*. Many computer animation programs incorporate the use of these fully jointed kinematic chains. Moving one link will result in the flexible movement of all the attached links.

The endpoints of the different links in the kinematic chain have special names. The *proximal end* is the fixed end of the first link in the chain. The *root* is a point at the proximal end that is the point of articulation of the link; thus, the root is a joint. The *distal end* is the end of the final link in the chain. The *end effector* (or just *effector*) is the point at the distal end that can be used to move the link and chain.

A *skeleton*, also known as an *armature*, is a hierarchy of articulated chains. A link in the hierarchy is said to be a *parent* link if there is a link or structure of links below it in the hierarchy. Similarly, a link is said to be the *child* of the link just above itself in the hierarchy. Each child can be transformed independently of other nodes in the hierarchy. Transformations applied to a parent propagate down to all of its children. Each level of the hierarchy has its own coordinate system and its own local origin. Objects rotate around their own center. A branch rotates around its parent's center. In mathematical terms, the kinematic chain is represented as a concatenated ordered sequence of translation, scale, and rotation, most typically calculated by using matrix algebra.

A *kinematic model* is a kinematic chain together with the geometry that surrounds the chain; the chain defines the relationship between the parts. There are two basic types of kinematic models: those that incorporate a rigid geometry and those that incorporate a flexible geometry. This flexible geometry, sometimes referred to as a *skin* or *envelope*, moves and deforms as the underlying chain moves.

1. Forward Kinematics

One of the most elementary ways to animate chains is to use *forward kinematics*, a technique whereby the animator specifies each joint angle for each pivot in an articulated chain or hierarchy, starting from the root of the chain and working downward. That is, the transformations being applied to the chain are applied to the root first and then work their way down to the distal end. The animator or computer animation program needs to determine all of the angles and positions. Typically, the specification of angles is not done for every frame, but only for the extreme positions, or key frames, with the computer used to calculate the in-between frames.

The advantage of this method is that the animator has full control of all the joints. The disadvantage is that there are a great many degrees of freedom to control, scripting is tedious, and if the angles are not controlled masterfully, the resulting animated action will not be convincing.

2. Inverse Kinematics

Since the animator has to specify all of the positions throughout the animation, forward kinematics becomes difficult to use once the models go beyond simple objects.

Inverse kinematics is a method that allows the animator to specify the position and orientation of the end effector. The animation program calculates all of the intermediate joint angles and then positions all of the individual pieces of the chain. The position of the end effector is specified in terms of a goal (for example, where the hand needs to be) and hence is also called *goal-directed animation*. Inverse kinematics concerns specifying a goal, and then computing automatically and correctly how to achieve a goal in a physically valid manner. The inverse kinematics approach, originated in the field of robotics, has the advantage of fixing positions and/or accelerations in advance, and letting the computer calculate the intermediate values.

It is noteworthy that no single solution to an inverse kinematics problem exists; there may be many different ways to position the intermediate joints to achieve the same goal position of the end effector. As long as a solution obeys the basic constraints of the mechanical system, it is a valid solution. The result is an envelope of permissible solutions. [Pocock, Rosebush 2002]

E. HUMANOID ANIMATION (H-ANIM) WORKING GROUP AND H-ANIM SPECIFICATIONS

H-Anim Working Group exists for the sole purpose of creating a standard representation for humanoids. H-Anim is a working group of the Web3D Consortium.

H-Anim's aim is to specify a way of defining interchangeable *humanoids* and *animations*. Animations include limb movements, facial expressions and lip synchronization with sound. One of the most important goals is to allow people to author humanoids and animations independently.

H-Anim Working Group published three specifications by the time this thesis was being written:

- H-Anim 1.0 Specification, http://h-anim.org/Specifications/H-Anim1.0/
- H-Anim 1.1 Specification, http://h-anim.org/Specifications/H-Anim1.1/
- H-Anim 200x Specification (Draft),

  http://www.h-anim.org/Specifications/H-Anim2001/

The most important difference between the H-Anim 200x specification and the others is that it will be independent of VRML97. However, H-Anim Working Group decided to describe the nodes using syntax familiar to the H-Anim community. In this manner everyone can examine the changes without being confused by the descriptions used. An overview of H-Anim 2001 Specification follows.

The human body consists of a number of segments (such as the forearm, hand and foot), which are connected to each other by joints (such as the elbow, wrist and ankle). In order for an application to animate a humanoid, it must obtain access to the joints and alter the joint angles. The application may also need to retrieve information about such elements as joint limits and segment masses.

A mesh of polygons typically defines each segment of the body, and an application may need to alter the locations of the vertices in that mesh. The application may also need to obtain information about which vertices are to be treated as a group for the purpose of deformation.

An H-Anim file contains a set of *joint* nodes that are arranged to form a hierarchy. Each joint node can contain other joint nodes and may also contain a *segment* node, which describes the body part associated with that joint. Each segment can also have a number of *site* nodes, which define locations relative to the segment. Sites can be used for attaching clothing and jewelry and can be used as end-effectors for inverse kinematics applications. They can also be used to define eye points and viewpoint locations.

Each Segment node can have a number of *displacer* nodes that specify which vertices within the segment correspond to a particular feature or configuration of vertices.

The file also contains a single *humanoid node*, which stores human-readable data about the humanoid such as author and copyright information. That node also stores references to all the joint, segment and site nodes, and serves as a "wrapper" for the humanoid. In addition, it provides a top-level Transform for positioning the humanoid in its environment. [H-Anim 2001]

The H-Anim Specification defines some abstractions for segments and joints to allow a human body to be described in a structured and standardized way. An H-Anim body is typically built as a series of nested joint nodes, each of which may have a segment associated with it. The Specification also provides naming conventions for joints and their associated segments. (see Figure 2.3)

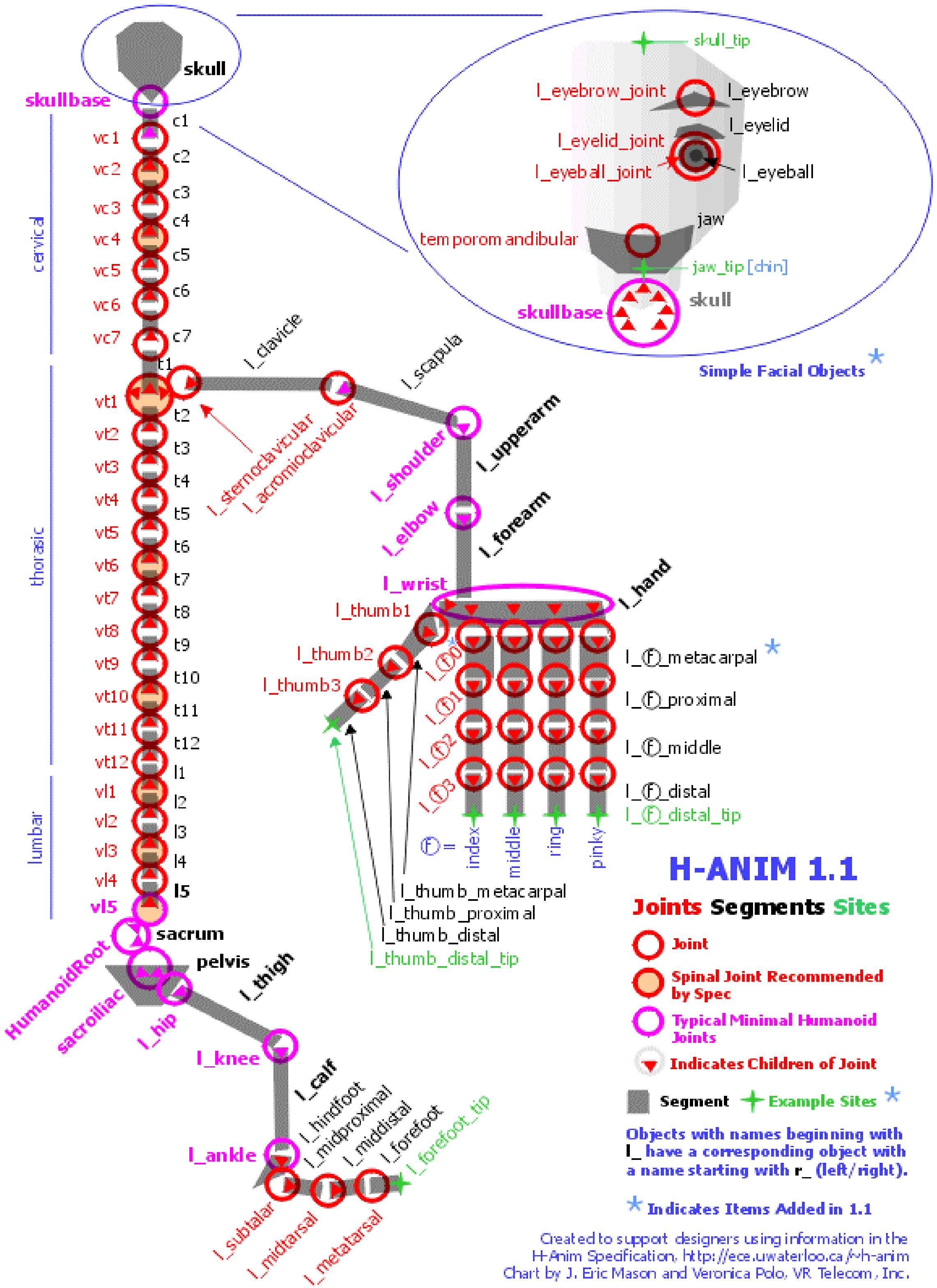


Figure 2.3 H-Anim 1.1 Specification Joint, Segment and Site Hierarchy [H-Anim 1.1]

F. HUMANOID MODELS

1. Nancy

Nancy is the canonical exemplar of H-Anim 1.1 Specification. The model was created by Cindy Ballreich, who grants permission for the use of Nancy for this project. Nancy.wrl consists of 2082 polygons and contains 17 joints, 15 segments and 4 viewpoints. It also contains a motion library with four behaviors: Stand, Walk, Run, Jump. Clicking on the appropriate text, which a Touch Sensor is attached, can activate these behaviors. (see Figures 2.4 – 2.7)

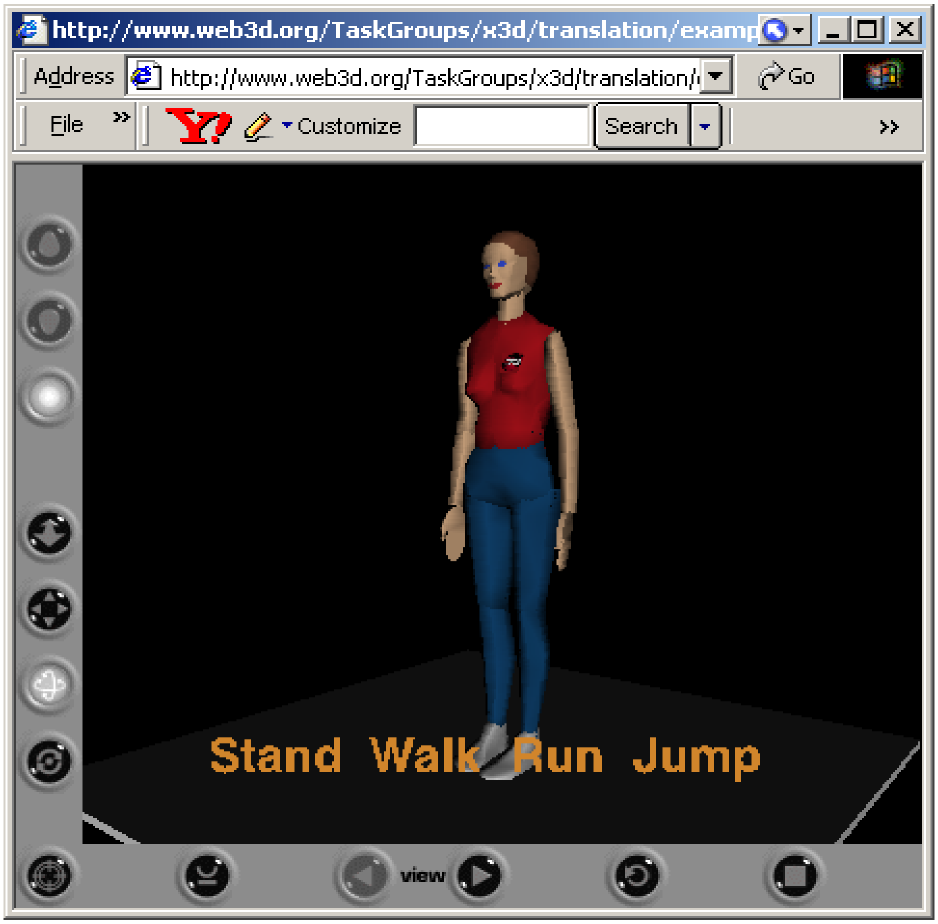

http://www.web3d.org/TaskGroups/x3d/translation/examp
Address
http://www.web3d.org/TaskGroups/x3d/translation/
Go
File
Customize
Search
Stand Walk Run Jump
view

Figure 2.4 Nancy Demonstrating the Stand Behavior

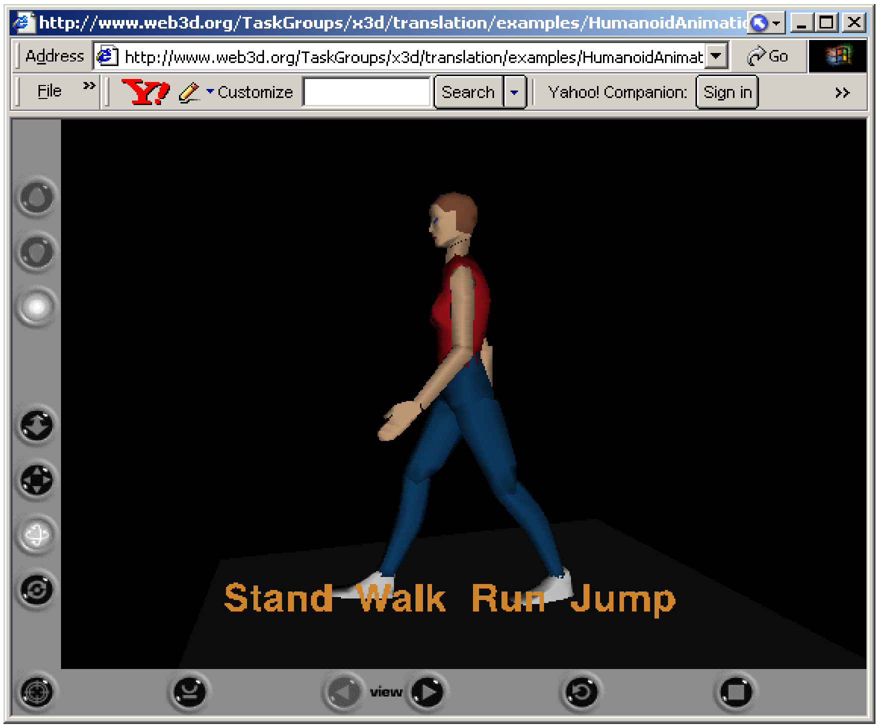

http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimatio
Address
http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimat
Go
File
Customize
Search
Yahoo! Companion:
Sign in
Stand
Walk
Run
Jump
view

Figure 2.5 Nancy Demonstrating the Walk Behavior

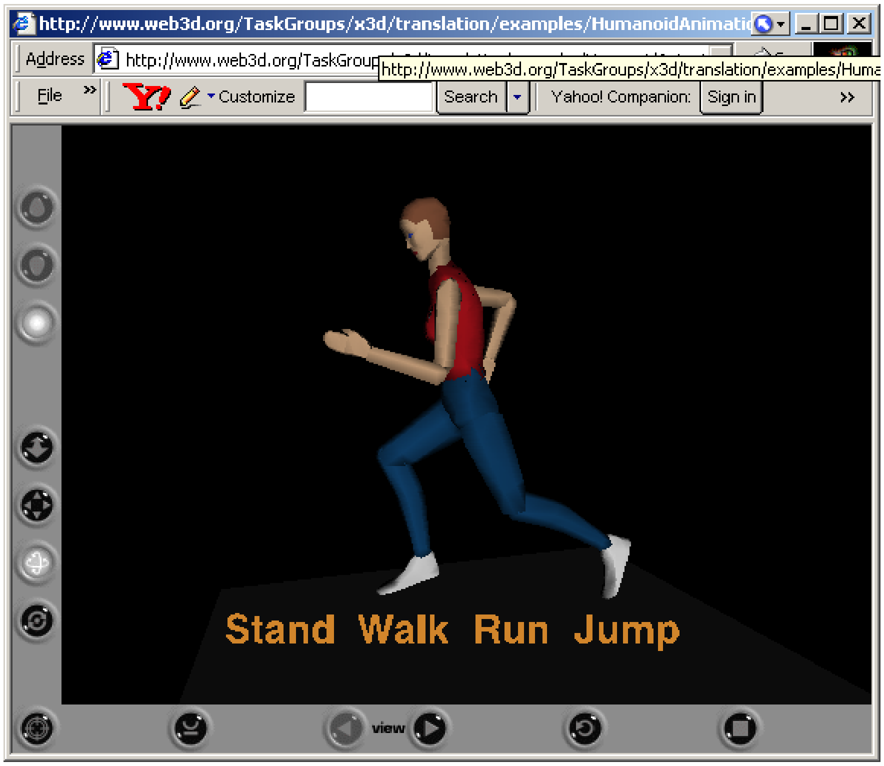

http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimatio
Address
http://www.web3d.org/TaskGroup
http://www.web3d.org/TaskGroups/x3d/translation/examples/Huma
File
Customize
Search
Yahoo! Companion:
Sign in
Stand
Walk
Run
Jump
view

Figure 2.6 Nancy Demonstrating the Run Behavior

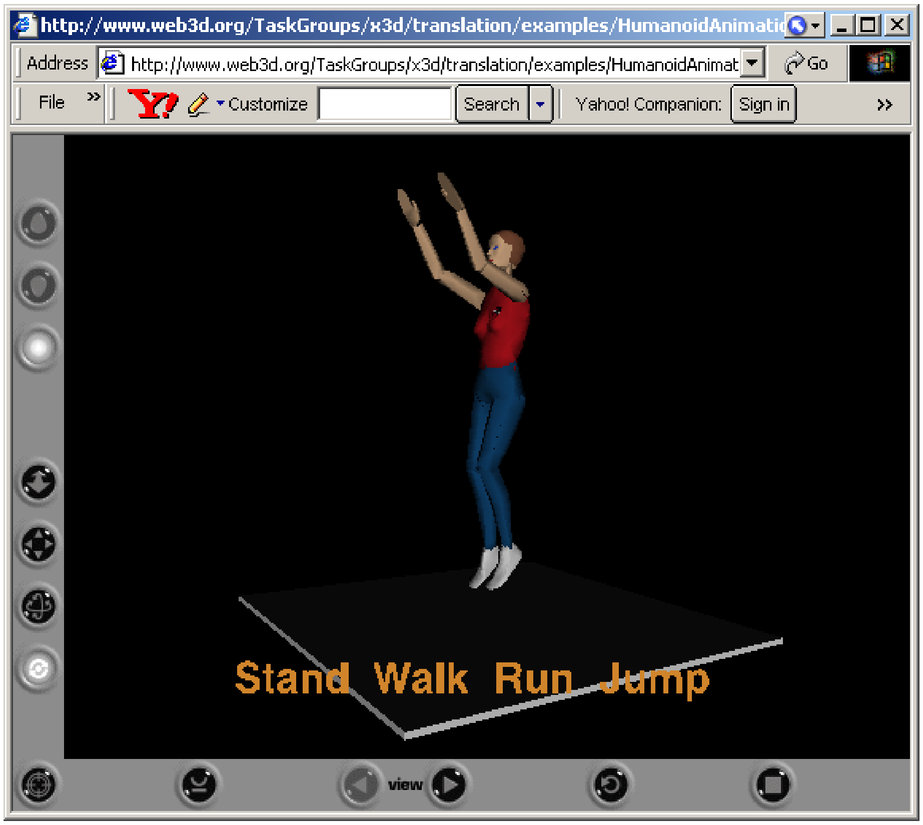

http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimati
Address
http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimat
Go
File
Customize
Search
Yahoo! Companion:
Sign in
Stand
Walk
Run
Jump
view

Figure 2.7 Nancy Demonstrating the Jump Behavior

2. Allen

Allen, created by Allen Dutton, consists of 10,000 polygons. It was initially a laser scan data cloud. At Naval Postgraduate School (NPS), its author converted it to a fully articulated, texture-mapped avatar that is capable of scripted movement [Dutton 2001]. The following steps were taken for this conversion:

- Polygon Reduction,
- Translating between File Formats (.ply to .wrl),
- Segmenting,
- Constructing the Avatar. Nancy was the foundation for the Allen.

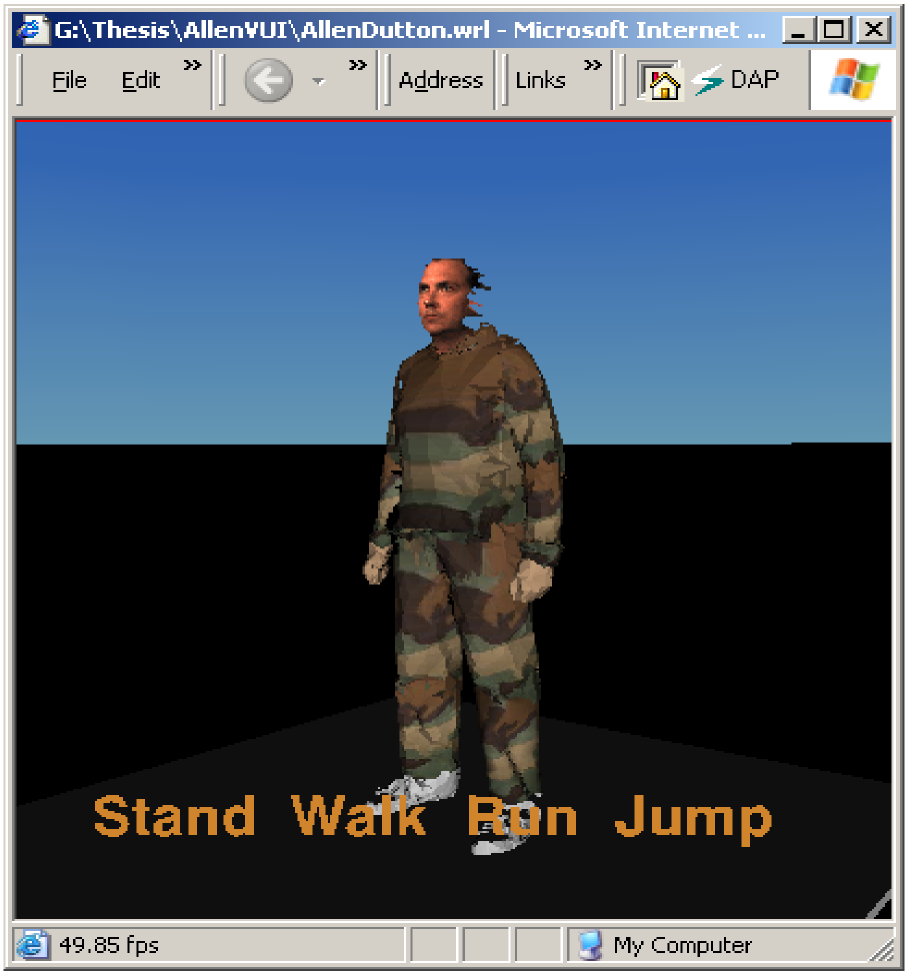


Figure 2.8 Allen; A Texture Mapped, Fully Articulated Avatar Converted from Laser Scan Data Cloud [Dutton 2001]

3. Box Man

Box Man, created by James Smith, is a seamless VRML Human demonstrating the H-Anim 2001 Specification. Seamless objects are the collection of polygons that consist of vertices, which do not lose their integrity even if a transformation is applied. A deformation engine provides this feature. Another significant difference is that Box Man has a skin, which Nancy and Allen do not have.

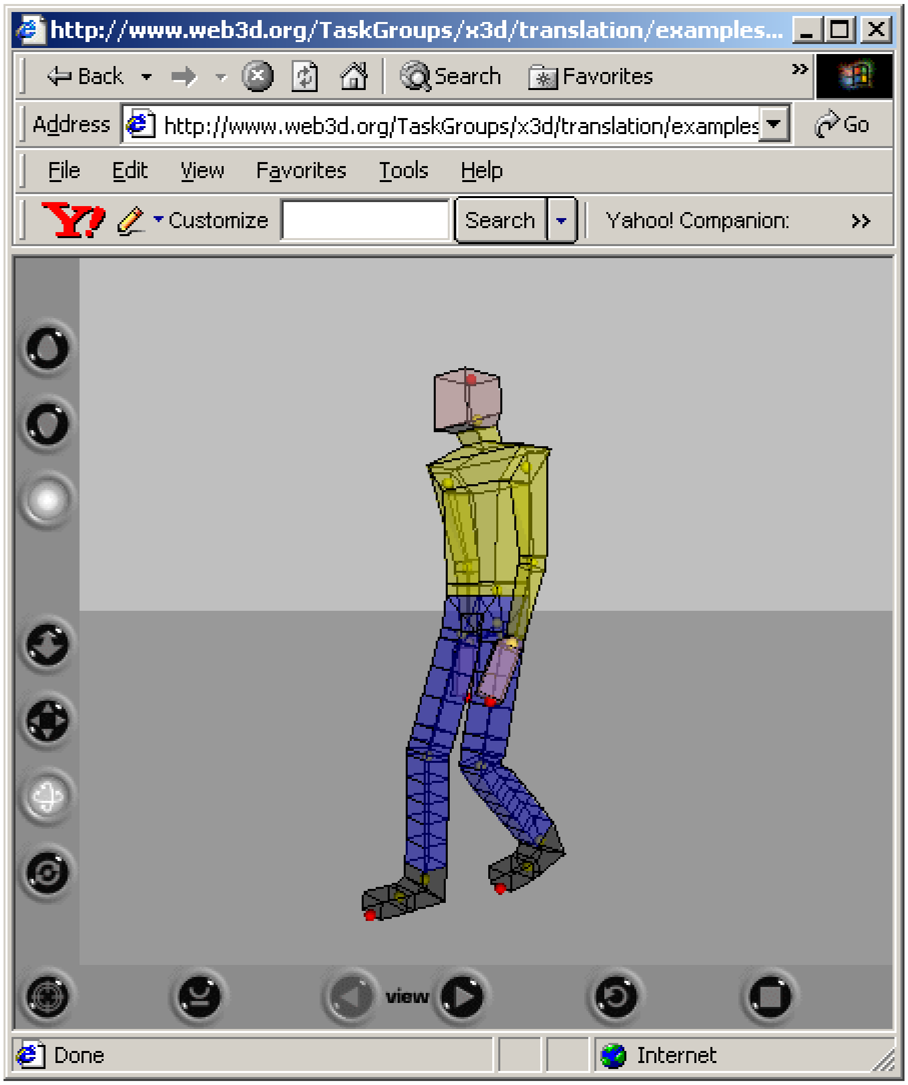

http://www.web3d.org/TaskGroups/x3d/translation/examples...
Back
Search
Favorites
Address
http://www.web3d.org/TaskGroups/x3d/translation/examples
Go
File
Edit
View
Favorites
Tools
Help
Customize
Search
Yahoo! Companion:
view
Done
Internet

Figure 2.9 Box Man; A Seamless Avatar

G. INTEGRATING VIRTUAL HUMANS INTO NETWORKED VURTUAL ENVIRONMENTS (Net-VEs)

Virtual humans are virtual organisms, which bring dynamism to virtual worlds. Providing an interface to aggregate and control articulated humans in a networked virtual environment (Net-VE) is a major issue. Tom Miller achieves this objective in his thesis [Miller 2000] by addressing the following areas:

- Virtual human avatars must have an articulated joint structure and at least a limited motion library in order to model realistic movement.

- A set of rule-based physical and logical behaviors for groups of humans must be developed and implemented in order to execute basic tactical formations and activities.
- Human entities must be able to aggregate into a group or mount other human entities (such as vehicles) and then separate back to individual entity control. Otherwise, the high-precision relative motion needed for group activities is not possible across network delays or in geo-referenced locations.

H. SUMMARY

This chapter discussed the Virtual Reality Modeling Language (VRML), Extensible 3D (X3D) and Kinematics. In addition, Humanoid Animation Working Group, Humanoid Animation Specifications and Humanoid Models were examined.

III. SPEECH-RECOGNITION TECHNOLOGY

A. INTRODUCTION

This chapter introduces the history, basic concepts of speech recognition and explains how the speech-recognition process works. Factors affecting speech-recognition accuracy follow.

B. WHAT MAKES EACH HUMAN VOICE DISTINCT?

This question is a significant one, if the amount of variations in a human voice is considered. Otolaryngology gives a physiological answer:

The vocal cords provide the sound source for speech. The length of the vocal cords determines the pitch of the voice; longer vocal cords produce lower-pitched (more masculine) tones. However, the vocal cords contribute only partially to the overall voice. In fact, outside the human body, the cords can only make a very unpleasant buzzing sound. Another important factor is the resonating chamber of the throat and nasal cavities. These cavities uniquely shape the sound of a person's voice for each individual, molding the buzzing sound from the larynx into a sound with character. Lastly, the muscles in the tongue, palate and lips provide the articulation to the voice. These articulator muscles determine elements, such as accent, lisp, or other distinctive speech patterns. When these factors are combined, a considerable amount of variation may influence a person's voice characteristics. [Simpson 99]

C. HISTORY

Interestingly, a toy company logged the first success story in the field of speech recognition many decades before major research in the area was considered. “Radio Rex” was a celluloid dog that responded to its spoken name. Lacking the computation power that drives recognition devices today, Radio Rex was a simple electromechanical device. [Maurer]

The dog was held within its house by an electromagnet. As current flowed through a circuit bridge, the magnet was energized. The bridge was sensitive to 500 cps of acoustic energy. The energy of the vowel sound of the word “Rex” caused the bridge to vibrate, breaking the electrical circuit, and allowing a spring to push Rex out of his house. While Radio Rex was not a commercial success, the toy was no doubt a pioneer in speech recognition.

The U.S. Department of Defense (D.O.D.) sponsored the first academic pursuits in speech recognition in the late 1940s. In an attempt to expedite the processing of intercepted Russian messages, the U.S. was eager to develop an automatic language translator. The first and most difficult step required to produce such a system was creating the ability to recognize speech. The project was a failure; typical results produced faulty translations, such as “the spirit is willing but the flesh is weak” into Russian and back into English as “the vodka is strong but the meat is disgusting.”

However, the D.O.D. recognized how much research was needed to achieve even a glimpse of success in speech recognition. As a result, the government funded the Speech Understanding Research (SUR) program at Carnegie Mellon University, MIT, and some select commercial institutions. The agency that funded the research later became known as the Defense Advanced Research Project Agency (DARPA).

In 1952, as the government-funded research began to gain momentum, Bell Laboratories developed an automatic speech recognition system that successfully identified the digits 0 to 9 when spoken over the telephone. Major developments at MIT followed. In 1959, a system successfully identified vowel sounds with 93% accuracy. Then seven years later, a system with a vocabulary of 50 words was successfully tested. In the early 1970s, the SUR program yielded its first substantial results. The HARPY system, at Carnegie Mellon University, could recognize complete sentences that consisted of a limited range of syntax. Nevertheless, the computing power it required was prodigious; it took 50 contemporary computers to process a recognition channel.

At this point, at least three key obstacles impeded a commercially viable product: computing power, the ability to recognize speech from any person (not just the particular voices the system has been designed around), and a continuity-of-speech capability (so that the person did…not…have…to…speak…with…constant…pauses…like…this).

Nonetheless, the successes of the 1950s and 1960s gained the attention of more and more commercial entities, and the most important goal of speech recognition became imaginable: Continuous Speech Recognition.

Companies devoted to commercializing speech recognition became more noticeable as the technology became more viable. Speechworks and Dragon Systems were two of the major companies that achieved vast reductions in the amount of processing power required by speech-recognition systems. This reduction in the need for processing power quickened the arrival of the crucial point at which the available processing power in computing systems equaled the processing power required by speech-recognition software. In addition, just as the processing power requirements were plummeting, so were the recognition-error rates. The combination of these effects set the stage for widespread commercial usage.

In 1996, Charles Schwab was the first major consumer company to implement a speech-recognition system for its vital customer interface. The system was called *Voice Broker*, and its success led to speech recognition being adopted by the likes of Sears, Roebuck and Co., United Parcel Service of America Inc., and E*Trade Securities. [NetByTel]

Technological innovations continued and in 1997, Dragon Systems introduced *Naturally Speaking* the first continuous speech dictation software available. In 2000, TellMe introduced the first global voice portal and later that year NetByTel launched the first voice enabler, which allowed users to fill out a web-based data form over the phone. [NetByTel] [Maurer]

D. BASIC TERMS AND CONCEPTS

Speech-recognition technology allows the user to provide input to an application with his or her own voice, just like typing on a keyboard or clicking a mouse. The software component, which performs the speech-recognition process, is called the *speech-recognition engine*. The speech-recognition engine processes the spoken input and translates it into text, which an application understands. If the application handles the recognized text simply as text and dictates the input, then it is considered a *dictation* *application*. An example of a dictation application is that the user says, “Open a new document,” and the application returns the text, “Open a new document.” If the application interprets the result of the recognition process as a command, then it is a *command and control application*. For instance, the user says, “Open a new document,” and the application opens a new document within a word processor program. Following are the basic terms and concepts that are fundamental to speech recognition.

1. Phonemes

Phonemes are the smallest sound units of which words are composed. A speech-recognition system stores a list of what phonemes sound like. That is, a table of relative formant positions is kept by the software, and the frequencies extracted from one’s speech are compared to this table. Phonemes can be placed in categories depending on the distinctive features they share. These categories are

a. Voiced Phonemes

Voiced phonemes consist of vowels and consonants, which use the vocal chords. i.e. “y,” “g” and “ng.”

b. Unvoiced Phonemes

Unvoiced phonemes consist of consonants, which don’t involve the vocal chords. i.e. “s” and “sh.”

c. Nasal Phonemes

Nasal phonemes consist of consonants such as “n” and “m,” which are produced in the nasal passages. [Kavanagh 95]

2. Acoustic Model

The acoustic model captures the acoustic properties of speech and provides the probability of the observed acoustic signal when given a hypothesized word sequence. The acoustic model can trace the differences in speech signals and then compare the features of these signals with known features of a language’s basic sounds in order to determine the spoken words. However, tracing the difference is not that simple, since a great deal of variety between any two acoustic signals is caused by factors such as different speakers, different emotions, and different speech rates.

3. Utterance

An utterance is any stream of speech between two periods of silence. Utterances are sent to the speech engine to be processed. Since silence delineates the start and end of an utterance, it is almost as important as what is spoken. When the speech-recognition engine detects audio input, the beginning of an utterance is signaled. Similarly, when the engine detects a certain amount of silence following the audio, the end of the utterance occurs. If the user does not say anything, the engine returns a silence timeout, which indicates that no speech was detected within the expected timeframe. In this case, the application may take an appropriate action, such as reprompting the user for input. An utterance can be a single word, or it can contain multiple words (a phrase or sentence). Whether these words and phrases are valid at a particular point in a dialog is determined by active grammars. If the user pauses too long between the words of a phrase, the end of an utterance can be detected too soon, and the engine will only process a partial phrase.

4. Pronunciation

The speech-recognition engine uses a great variety of disparate data, statistical models, and algorithms to convert spoken into text. One piece of information that the speech engine uses to process a word is its pronunciation, which represents what the speech engine predicts a word will sound like. Words can have multiple pronunciations. The application developer may want to provide multiple pronunciations for certain words and phrases to allow variations in the ways user may speak them.

5. Grammar

The allowed order of words and phrases that users can say to the speech-recognition application must be specified. These words and phrases are defined to the speech-recognition engine and are used in the recognition process. A grammar uses a particular syntax, or set of rules to define the words and phrases that can be recognized by the engine. A grammar can be as simple as a list of words, or it can be flexible enough to allow such variability in what can be said that it approaches natural language capability. Grammars define the domain, or context, within which the recognition engine works. The engine compares the current utterance against the words and phrases in the active grammars. If the user says something that is not in the grammar, the speech engine will not be able to decipher it correctly. The design of application grammars needs to be thought out carefully. They can be as restrictive or as flexible as the user and application need them to be. Since there are tradeoffs between recognition speed (response time) and accuracy versus the size of the grammar(s), an application designer may experiment with different grammar designs to validate one that best matches the users' requirements and expectations.

6. Natural Language Commands

Natural Language Commands are aimed at providing a more intuitive way of using a speech-recognition application. Rather than having to speak commands by reading from menus, the user can informally state the desired actions and the software intelligently interprets the instructions to perform the task. This technology is still in its infancy and is limited in what applications and actions it can support, but it holds great promise. [PCMagazine 99]

An unrestricted natural language interface is generally considered an enticing prospect because, if it could be implemented, it would offer many advantages: it would be easy to learn and easy to remember, because its structure and vocabulary are already familiar to the user; because the same language could be used for many application, there might be fewer transfer problems between applications; they are particularly powerful because of the multitude of ways in which to accomplish an action; and they also allow considerable flexibility in executing the steps of a task. [Long 94]

7. Training

Most of the recognition applications require an initial training and enrollment process in order to teach the software to recognize the user's voice. A voice profile that is unique to that individual is then produced. This procedure also helps the user learn how to speak to a computer. After the initial training, the program's accuracy will improve as the user dictates by correcting the mistakes in transcription. Another way the program improves in accuracy is by analyzing existing documents for new words and for the user's syntax. These three steps–initial training, making corrections, and vocabulary analysis–are common for almost all dictation applications. [Fulton 2000]

8. Speaker Dependence vs. Speaker Independence

Speaker dependence describes the degree to which a speech-recognition system requires knowledge of a speaker's individual voice characteristics to process speech. The speech-recognition engine can learn how the user speaks words and phrases; it can be trained to recognize the user's voice. Speech-recognition systems that require a user to train the system to his or her voice are known as *speaker-dependent* systems. Most dictation applications are speaker dependent. Because they operate on very large vocabularies, dictation applications perform much better when the speaker has spent several hours to train the system to his or her voice.

Speech-recognition systems that do not require a user to train the system are known as *speaker-independent* systems. These systems successfully process the speech of many different users without having to understand each individual's voice characteristics.

9. Accuracy

The most widely used measurement to define the performance of a speech-recognition system is accuracy, typically a quantitative measurement that can be calculated in several ways. The most important measurement of accuracy is whether the desired end result occurred, which is useful in validating application design. For example, if the user says "yes," the engine returns "yes," and the "yes" action is executed, then desired end result is clearly achieved. Nevertheless a condition in which the text returned by the engine does not exactly match the utterance may occur. For instance, what if the user says "nope," the engine returns "no," yet the "nope" action is executed? Should this dialog be considered successful? This example is successfully accurate, because mostly due to choice of grammar the desired end result is achieved.

Another measurement of recognition accuracy is whether the engine recognized the utterance exactly as spoken. This measure of recognition accuracy is expressed as a percentage and represents the number of utterances recognized correctly out of the total number of utterances spoken. Using the previous example, if the engine returns “no” when the user says “nope,” this is considered a recognition error. Based on the accuracy measurement, the grammar may be analyzed to improve accuracy.

E. HOW IT WORKS

Basic terms and concepts of speech-recognition technology are presented in previous sections. These terms are now linked to show how the speech-recognition process works.

A microphone converts the user’s voice into an analog signal and feeds it to the PC’s sound card. An analog-to-digital converter takes the signal and converts it to a stream of digital data. The software receives the digital data and processes it. The acoustic model removes noise and unneeded information such as changes in volume. Using mathematical calculations, the model reduces the data to a spectrum of frequencies (the pitches of the sound), analyzes the data, and converts the words into digital representations of phonemes. The speech-recognition engine has a rather complex task to handle, namely taking raw audio input and translating it to a recognized text, which an application understands. Once the data is in the proper format, the engine searches for the best match. The engine does this by considering the words and phrases it recognizes (i.e. the active grammars), along with its knowledge of the environment in which it is operating. The knowledge of the environment is provided in the form of an acoustic model. Once the engine identifies the most likely match for what was said, it returns a result as a string. Most speech engines try very hard to find a match and are usually very forgiving. But it is important to note that the engine always returns its best guess.

The result, which the recognition engine returns, can be either of two states: *acceptance* or *rejection*. The engine flags acceptance or rejection with each processed utterance. An accepted utterance is one in which the engine returns recognized text. Sometimes the match may be poor because the user said something that the application cannot accept, or the user spoke indistinctly. In these cases, the speech engine returns the closest match, which might be incorrect. Some engines also return a confidence score along with the text to indicate the likelihood that the returned text is correct. Figure 3.1 illustrates the major components of speech recognition.

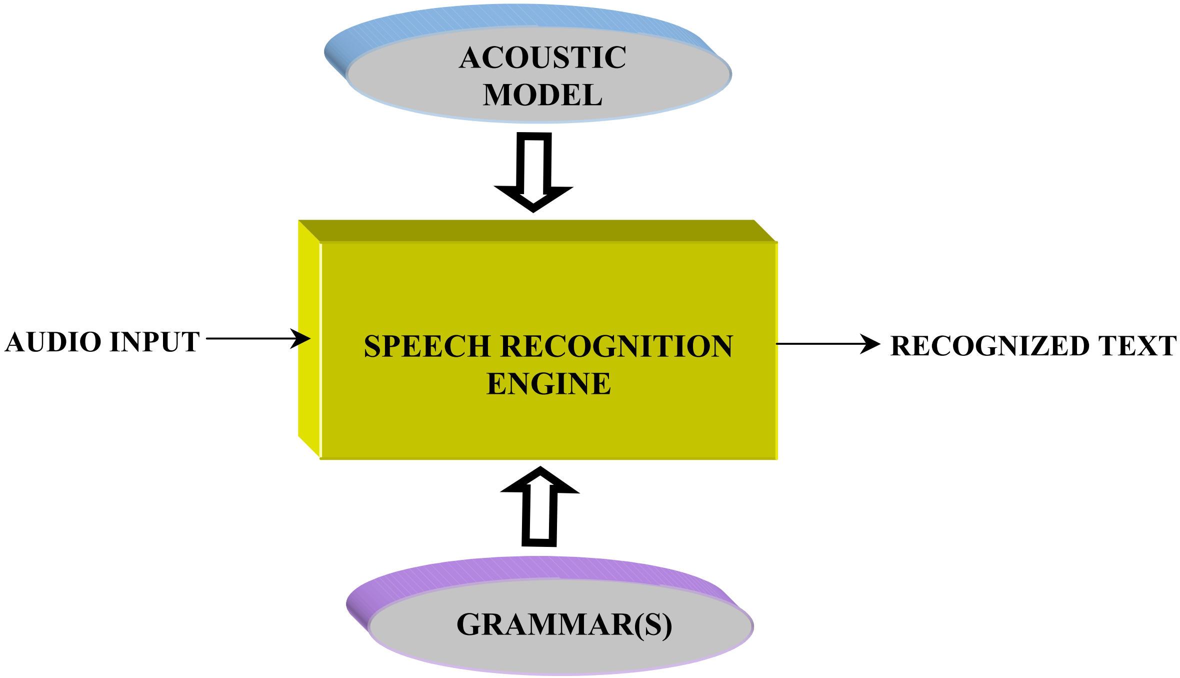


Figure 3.1 Major Components in a Speech-Recognition System [Kemble 2001]

F. FACTORS AFFECTING THE ACCURACY

Many factors affect speech-recognition accuracy. These factors can be categorized as:

- *Environment:* Background noise can influence the recognition process. The human auditory system can reduce the effect of noise on speech perception. Speech-recognition technology cannot block out unwanted noise initially, but it can remove it from the data layer. Systems can perform with very low error rates as long as the environmental conditions remain controlled, constant and quiet. Performance degrades when noise is added to the scenario or when the environment differs from the training session used to make the reference templates.

- *Hardware:* Computers set up to use speech-recognition software must be on the leading edge rather than the trailing edge. They must be fast and have expansive memory (cache and RAM). Sound cards and microphone quality are further factors.
- *Speaker / User:* The clarity and naturalness of the speaker's annunciation significantly affects the accuracy. Users with accents or atypical voices may result in lower accuracy. Single-user input is easier to recognize than speech from multiple speakers because most representations of speech are sensitive to characteristics of the speaker. If there is variety of speakers, the pattern-matching templates and models for one person might not perform as well for some individuals as for others. To improve accuracy, training is essential in many speech-recognition applications. These applications require the user to train with the system so the program becomes accustomed to the user's unique voice characteristics.
- *Vocabulary Size:* A vocabulary is the collection of words that the pattern-matching algorithm knows and compares the input against. Larger vocabularies are more likely to contain ambiguous words than smaller vocabularies. Ambiguous words are those with similar pattern-matching templates. These words can confuse the recognition algorithms. Also, when the vocabulary size increases, searching the speech-model database takes longer, thus software and application designers have to weigh faster response time against higher recognition accuracy when expanding the allowed vocabulary.
- *Grammar:* The grammar of the recognition domain defines the allowable sequence of words. A tightly constrained grammar is one in which the number of words that can follow any certain word is small. The amount of

constraint on word choice is referred to as the *perplexity* of the grammar. Systems with low perplexity are potentially more accurate than those that give the user more freedom because the system limits the vocabulary and the search space to those words that can occur according to the current context.

G. SUMMARY

This chapter provides background information about the history and basic concepts of speech recognition. Operation of the speech-recognition process and factors affecting the accuracy are explained.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. THE JAVA SPEECH API

A. INTRODUCTION

This chapter examines the Java Speech API and describes the speech technologies that it supports.

B. OVERVIEW

The Java Speech API, developed by Sun Microsystems in cooperation with speech technology companies, defines a software interface that allows developers to take advantage of speech technology for personal and enterprise computing [JSAPI 1.0]. By leveraging the inherent strengths of the Java platform, the Java Speech API enables developers of speech-enabled applications to incorporate more sophisticated and natural user interfaces into Java applications and applets that can be deployed on a wide range of platforms.

The Java Speech API defines a standard, cross-platform software interface to state-of-the-art speech technology. Two core speech technologies are supported through the Java Speech API: *speech recognition* and *speech synthesis*. Speech recognition allows computers to listen to spoken language and to determine what has been said. In other words, speech-recognition processes audio input containing speech by converting it into text. Speech synthesis provides the reverse process of producing synthetic speech from text provided by an application, an applet or a user. Speech synthesis is often referred to as *text-to-speech* (TTS) technology.

Speech interfaces give Java application developers the opportunity to implement distinct and engaging personalities for their applications and to differentiate their products. The Java Speech API is an extension to the Java platform. Extensions are packages of classes written in the Java programming language (and any associated native code) that application developers can use to extend the functionality of the core part of the Java platform.

The design goals for the Java Speech API were:

- Provide support for speech synthesizers and for both command-and-control and dictation speech recognizers,
- Provide a robust cross-platform, cross-vendor interface to speech synthesis and speech recognition,
- Enable access to state-of-the-art speech technology,
- Support integration with other capabilities of the Java platform, including the suite of Java Media APIs.

C. CORE SPEECH TECHNOLOGIES

1. Speech Synthesis

A speech synthesizer converts written text into spoken language. Speech synthesis is also referred to as *text-to-speech* (TTS) conversion.

The major steps in producing speech from text are as follows:

- *Structure Analysis*: Process the input text to determine where paragraphs, sentences and other structures start and end. For most languages, punctuation and formatting data are used in this stage.
- *Text Pre-Processing*: Analyze the input text for special constructs of the language. In English, special treatment is required for abbreviations, acronyms, dates, times, numbers, currency amounts, e-mail addresses and many other forms. Other languages need special processing for these forms and most languages have other specialized requirements.

The result of these first two steps is a spoken form of the written text. The following table (Table 4.1) demonstrates examples of the difference between written and spoken text.

Written Text (Before Pre-Processing)	**Spoken Text (After Pre-Processing)**
Leave at 6:30 on 6/15/99.	"Leave at six thirty on June fifteenth nineteen ninety nine."
Add $50 to account 69243.	"Add fifty dollars to account six nine, two four three."

Table 4.1 Written and Spoken Text Examples

The remaining steps convert the spoken text to speech.

- *Text-to-Phoneme Conversion*: Convert each word to *phonemes*. US English has around 45 phonemes including the consonant and vowel sounds. For example, "times" is spoken as four phonemes "t ay m s." Different languages have different sets of sounds (different phonemes). For example, Japanese has fewer phonemes, including sounds not found in English, such as "ts" in "tsunami."

- *Prosody Analysis*: Process the sentence structure, words and phonemes to determine the appropriate *prosody* for the sentence. Prosody includes many of the features of speech other than the sounds of the words being spoken. This includes the pitch (or melody), the timing (or rhythm), the pausing, the speaking rate, the emphasis on words and many other features. Correct prosody is important for making speech sound right and for correctly conveying the meaning of a sentence.

- *Waveform Production*: Finally, the phonemes and prosody information are used to produce the audio waveform for each sentence. The speech can be produced from the phoneme and prosody information in many ways. Most current systems achieve this in one of two ways: *concatenation* of pieces of recorded human speech, or *formant synthesis* using signal processing techniques based on knowledge of how phonemes sound and how prosody affects those phonemes.

a. Speech-Synthesis Limitations

Speech synthesizers can make errors in any of the processing steps described above. Human ears are well-tuned to detecting such errors, so careful work by developers is needed to minimize errors and improve the speech-output quality.

b. Speech-Synthesis Assessment

The major feature of a speech synthesizer that affects its understandability, its acceptance by users, and its usefulness to application developers is output quality. Knowing how to evaluate speech synthesis quality and knowing the factors that influence the output quality are important when deploying speech synthesis.

Humans are conditioned by a lifetime of listening and speaking. The human ear and brain are highly sensitive to small changes in speech quality. A listener can detect changes that might indicate a user's emotional state, an accent, a speech problem or many other factors. The quality of current speech synthesis remains far below that of human speech, so listeners must make more effort than normal to understand synthesized speech and must ignore errors. For new users, listening to a speech synthesizer for extended periods can be tiring and unsatisfactory.

The two key factors a developer must consider when assessing the quality of a speech synthesizer are its *understandability* and its *naturalness*. Understandability indicates how reliably a listener will understand the words and sentences spoken by the synthesizer. Naturalness indicates the extent to which the synthesizer sounds like a human voice–a characteristic that is desirable for most applications, but not for all.

Understandability is affected by a speech synthesizer’s ability to perform all the processing steps described previously in combination, because any error by the synthesizer can potentially mislead a listener. Naturalness is affected more by the later stages of processing, particularly the processing of prosody and the generation of the speech waveform.

Although the concept might seem counter-intuitive, creating an artificial-sounding voice that is highly understandable is possible. Similarly, having a voice that sounds natural but is not always easy to understand is also possible.

2. Speech Recognition

Speech recognition is the process of converting spoken language to written text or some similar form. The basic characteristics of a speech recognizer supporting the Java Speech API are

- It is monolingual; it supports a single specified language.
- It processes a single input audio stream.
- It can optionally adapt to the voice of its users.
- Its grammars can be dynamically updated.
- It has a small, defined set of application-controllable properties.

The major steps of a typical speech recognizer follow:

- *Grammar Design*: Recognition grammars define the words that may be spoken by a user and the patterns in which the words may be spoken. A grammar must be created and activated for a recognizer to know how to listen for incoming audio.
- *Signal Processing*: Analyzes the spectrum (frequency) characteristics of the incoming audio.
- *Phoneme Recognition*: Compares the spectrum patterns to the patterns of the phonemes of the language being recognized.
- *Word Recognition*: Compares the sequence of likely phonemes against the words and patterns of words specified by the active grammars.
- *Result Generation*: Provides the application with information about the words the recognizer has detected in the incoming audio. The result

> information is always provided once recognition of a single utterance is complete, but this information may also be provided during the recognition process. The result always indicates the recognizer's best guess of what a user said, but may also indicate alternative guesses.

The primary way in which an application controls the activity of a recognizer is through control of its *grammars*. A grammar is an object in the Java Speech API, which indicates what words a user is expected to say and in what patterns those words may occur. Grammars are important to speech recognizers because they constrain the recognition process. These constraints make recognition faster and more accurate because the recognizer does not have to check for bizarre sentences.

The Java Speech API supports two basic grammar types: *rule grammars* and *dictation grammars*. These grammar types differ in the way in which applications set up the grammars, the types of sentences they allow, the way in which results are provided, the amount of computational resources required, and the way in which they are effectively used in application design. Other speech-recognizer controls available to a Java application include pausing and resuming the recognition process, direction of result events and other events relating to the recognition processes, and control of the recognizer's vocabulary.

In a rule-based speech-recognition system, an application provides the recognizer with rules that define what the user is expected to say. These rules constrain the recognition process. Careful design of the rules, combined with a careful user-interface design, can produce rules that allow users reasonable freedom of expression while still limiting the range of what may be said. In this manner, the recognition process is as fast and accurate as possible. Any speech recognizer that supports the Java Speech API must support rule grammars. The Java Speech Grammar Format Specification defines the full behavior of rule grammars, and also discusses how complex grammars can be constructed by combining smaller grammars.

Dictation grammars impose fewer restrictions on what can be said, making them closer to providing the ideal of free-form speech input. The cost of this greater freedom is that they require more substantial computing resources, require higher quality audio input, and tend to make more errors.

A dictation grammar is typically larger and more complex than rule-based grammars. Dictation grammars are typically developed by statistical training on large collections of written text. Fortunately, developers do not need to know any of these details because a speech recognizer that supports a dictation grammar through the Java Speech API has a built-in dictation grammar. An application that needs to use that dictation grammar simply requests a reference to it and enables it when the user might say something matching the dictation grammar.

Dictation grammars may be optimized for particular kinds of text. Often a dictation recognizer may be available with dictation grammars for general-purpose text, for legal text, or for various types of medical reporting. In these different domains, different words are used, and the patterns of words also differ.

A dictation recognizer in the Java Speech API supports a single dictation grammar for a specific domain. The application (or user, or both) select an appropriate dictation grammar when the dictation recognizer is selected and created.

a. Speech-Recognition Limitations

The two primary limitations of current speech-recognition technology are inability to robustly transcribe free-form speech input, and errors in accuracy. Most recognition errors fall into the following categories:

- *Rejection*: The user speaks but the recognizer cannot understand what was said. The outcome is that the recognizer does not produce a successful recognition result. In the Java Speech API, applications receive an event that indicates the rejection of a result.
- *Misrecognition*: The recognizer returns a result with words that are different from those that the user spoke. This is the most common type of recognition error.
- *Misfire*: The user does not speak, but the recognizer returns a result.

Table 4.2 lists some of the common causes of the three types of recognition errors.

PROBLEMS	POSSIBLE CAUSES
Rejection or Misrecognition	User speaks one or more words not in the vocabulary.
	User's sentence does not match any active grammar.
	User speaks before system is ready to listen.
	Words in active vocabulary sound alike and are confused (e.g., "too," "two.")
	User pauses too long in the middle of a sentence.
	User speaks with a disfluency (e.g., restarts sentence, stumbles, "umm," "ah.")
	User's voice trails off at the end of the sentence.
	User has an accent or cold.
	User's voice is substantially different from stored "voice models" (often a problem with children).
	Computer's audio is not configured properly.
	User's microphone is not properly adjusted.

Misfire	Non-speech sound (e.g., cough, laugh).
	Background speech triggers recognition.
	User is talking with another person.

Table 4.2 Speech-Recognition Errors and Possible Causes [From JSAPI 1.0]

b. Speech-Recognition Assessment

Speech recognizers make mistakes. So do people, but recognizers usually make more. Understanding why recognizers make mistakes, the factors that lead to these mistakes, and how to train users of speech recognition to minimize errors are important concepts for speech application developers.

The reliability of a speech recognizer is most often defined by its *recognition accuracy*. Accuracy is usually given as a percentage and is most often the percentage of correctly recognized words. Because the percentage can be measured differently and depends greatly upon the task and the testing conditions, comparing recognizers simply by their percentage recognition accuracy is not always possible. A developer must also consider the seriousness of recognition errors: misrecognition of a bank account number or the command "delete all files" may have serious consequences. The following is a list of major factors that influence recognition accuracy.

- Recognition accuracy is usually higher in a quiet environment.
- Higher-quality microphones and audio hardware can improve accuracy.
- Users that speak clearly (but naturally) usually achieve better accuracy.
- Users with accents or atypical voices may obtain lower accuracy.

- Applications with simpler grammars typically achieve better accuracy.
- Applications with less *confusable* grammars typically attain better accuracy. Similar-sounding words are harder to distinguish.

While these factors can all be significant, their impact can vary between recognizers because each speech recognizer optimizes its performance by trading off various criteria. For example, some recognizers are designed to work reliably in high-noise environments (e.g. factories and mines) but are restricted to very simple grammars. Dictation systems have complex grammars but require good microphones, quieter environments, and clearer speech from users and more powerful computers. Some recognizers adapt their process to the voice of a particular user to improve accuracy, but may require training by the user. Thus, users and application developers often benefit by selecting an appropriate recognizer for a specific task and environment.

Only some of these factors can be controlled programmatically. The primary application-controlled factor that influences recognition accuracy is grammar complexity. Recognizer performance can degrade as grammars become more complex and can degrade as more grammars are active simultaneously. However, making a user interface more natural and usable sometimes requires the use of more complex and flexible grammars. Thus, application developers often need to consider a trade-off between increased usability with more complex grammars and the decreased recognition accuracy this might cause.

3. Speech Engine

The `javax.speech` package of the Java Speech API defines an abstract software representation of a *speech engine,* which is the generic term for a system designed to deal with either speech input or speech output. Speech synthesizers and speech recognizers are both speech engine instances. Speaker-verification systems and speaker-identification systems are also speech engines but are not currently supported through the Java Speech API.

The `javax.speech` package defines classes and interfaces that define the basic functionality of an engine. The `javax.speech.synthesis` package and `javax.speech.recognition` package extend and augment the basic functionality to define the specific capabilities of speech synthesizers and speech recognizers.

The Java Speech API makes only one assumption about the implementation of a JSAPI engine: that it provides a true implementation of the Java classes and interfaces defined by the API. In supporting those classes and interfaces, an engine may be completely software-based or may be a combination of software and hardware. The engine may be local to the client computer or remotely operating on a server. The engine may be written entirely as Java software or may be a combination of Java software and native code.

The basic processes for using a speech engine in an application are as follows:

- Identify the application's functional requirements for an engine (e.g, language or dictation capability),
- Locate and create an engine that meets those functional requirements,
- Allocate the resources for the engine,
- Set up the engine,
- Begin operation of the engine-technically, resume it
- Use the engine,
- De-allocate the resources of the engine.

Applications are responsible for determining their functional requirements for a speech synthesizer and/or speech recognizer. For example, an application might determine that it needs a dictation recognizer for the local language or a speech synthesizer for Korean with a female voice. Applications are also responsible for determining behavior when there is no speech engine available with the required features. Based on specific functional requirements, a speech engine can be selected, created, and started.

Functional requirements are handled in applications as *engine selection properties*. Each installed speech synthesizer and speech recognizer is defined by a set of properties. An installed engine may have one or many *modes of operation*, each defined by a unique set of properties and encapsulated in a *mode descriptor* object.

The basic functionality provided by a synthesizer is speaking text, managing a queue of text to be spoken and producing events as these functions proceed. The Synthesizer interface extends the Engine interface to provide this functionality.

The basic functionality provided by a recognizer includes managing the grammar and producing results when a user makes utterances that match active grammars. The recognizer interface extends the engine interface to provide this functionality.

D. JAVA SPEECH GRAMMAR FORMAT (JSGF)

Speech-recognition systems provide computers with the ability to listen to user speech and determine what is said. Current technology capabilities do not yet support *unconstrained* speech recognition, i.e. the ability to listen to any speech in any context and transcribe it accurately. To achieve reasonable recognition accuracy and response time, current speech recognizers constrain what they listen for by using *grammars*.

The *Java Speech Grammar Format* (JSGF) defines a platform-independent means of describing one type of grammar, a *rule grammar* (also known as a *command and control grammar* or *regular grammar*). JSGF uses a textual representation that both developers and computers can read and can edit and can be included in the Java source code.

A rule grammar specifies the types of *utterances* a user might say (a spoken utterance is similar to a written sentence). For example, a simple window control grammar might listen for "open a file," "close the window," and similar commands. What the user can say depends upon the context: Desired results may vary greatly, depending on whether the user controlling an email application, reading a credit card number, or selecting a font (for example). Applications know their own context based on their operations, so applications are thus responsible for providing a speech recognizer with appropriate and corresponding grammars.

1. Definitions

a. Grammar Names and Package Names

Each grammar defined by Java Speech Grammar Format has a unique name that is declared in the grammar header. The *Full grammar name* is in the form of package name + simple grammar name, for instance: `com.sun.speech.apps.numbers.` A *simple grammar name* includes the grammar name only, for example: `numbers`. The package name and grammar name have the same format as packages and classes in the Java programming language.

b. Rulenames

A grammar is composed of a set of rules that together define what may be spoken. Rules are combinations of speakable text and references to other rules. Each rule has a unique *rulename*. A reference to a rule is represented by the rule's name in surrounding <> characters (less-than and greater-than).

Grammar developers should be aware of two specific constraints. First, rulenames are compared with exact Unicode string matches, so case is significant. For example, <Name>, <NAME> and <name> are different. Second, white space is not permitted in rulenames.

The rulenames <NULL> and <VOID> are reserved. <NULL> defines a rule that is automatically matched: that is, matched without the user speaking any words. <VOID> defines a rule that can never be spoken. Inserting <VOID> into a sequence automatically makes that sequence unspeakable.

c. Tokens

A *token*, sometimes called a *terminal symbol*, is the part of a grammar that defines what may be spoken by a user. Most often, a token is equivalent to a word. In Java Speech Grammar Format (JSGF), a token is a character sequence bounded by whitespace, by quotes or delimited by the other symbols that are significant in the grammar. A token is a reference to an entry in a recognizer's vocabulary, often referred to as the *lexicon*. The recognizer's vocabulary defines the pronunciation of the token. With such pronunciation, the recognizer is able to listen for that token.

A token does not need to be a single word. A token may be a sequence of words or a symbol. Quotes can be used to surround multi-word tokens and special symbols, for example, the "New York" subway.

d. Comments

Comments may appear in both the header and body. The comment style of the Java Programming Language is adopted. Comments may appear anywhere in a grammar definition, except within tokens, quoted tokens, rulenames, tags and weights.

e. Grammar Header

A single file defines a single grammar. The definition grammar contains two parts: the *grammar header* and the *grammar body*. The grammar header includes a self-identifying header, declares the name of the grammar and declares imports of rules from other grammars.

f. Grammar Body

The grammar body defines rules. Each rule is defined in a rule definition. A rule is defined once in a grammar. The order of definition of rules is not significant.

2. Rule Expansions

The simplest rule expansions are a reference to a token and a reference to a rule. For example,

```
<a> = horse;

<b> = <a>;

<c> = <com.acme.grammar.zenith>;
```

The rule <a> expands to a single token "horse." Thus, to speak <a> the user must say the word "horse." The rule <b> expands to <a>. This means that to speak <b>, the user must say something that matches the rule <a>. Similarly, to speak rule <c> the user must speak something that matches the rule <com.acme.grammar.zenith>.

3. Defining Complex Rules

Complex rules can be defined by logical combinations of legal expansions using the following procedures:

a. Composition and Sequences

A rule may be defined by a sequence of expansions. A sequence of legal expansions, each separated by white space, is itself a legal expansion.

<where> = I live in Monterey;

<statement> = this <object> is <Condition>;

To speak a sequence, each item in the sequence must be spoken in the defined order. In the first example, to say the rule <where>, the speaker must say the words, “I live in Monterey” in that exact order. The second example mixes tokens with references to the rules <object> and <Condition>. To say the rule <statement>, the user must say “this” followed by something that matches <object>, then “is,” and finally something matching <Condition>.

A rule may be defined as a set of alternative expansions separated by vertical bar characters ‘|’ and optionally by whitespace. For example:

<name> = Don | Nancy | Allen | Ozan | <otherNames>;

To say the rule <name>, the speaker must say one, and only one of the items in the set of alternatives. For example, a speaker could say, "Don," "Nancy," "Allen," "Ozan" or anything that matches the rule <otherNames>. However, the speaker could not say "Nancy Allen" because the | operator corresponds to exclusive or.

It is worthy noting that that Sequences have higher precedence than alternatives.

b. Grouping

Any legal expansion may be explicitly grouped using matching parentheses ‘()’. Grouping has a high precedence and so can be used to ensure the correct interpretation of rules. Grouping is also useful for improving clarity. For example, because sequences have higher precedence than alternatives, parentheses are required in the following rule definition so that "please close" and "please delete" are legal combinations.

```
<action> = please (open | close | delete);
```

Square brackets may be placed around any rule definition to indicate that the contents are optional. In other respects, they are equivalent to parentheses for grouping and has the same precedence. For example,

```
<polite> = please | oh mighty computer;

public <command> = [ <polite> ] don't crash;
```

The rule <command> allows a user to say, "Don't crash" and to optionally add one form of politeness such as, "Oh mighty computer, don't crash" or "Please, don't crash."

c. Unary Operators

There are three unary operators in the Java Speech Grammar Format: the Kleene star (*) operator, the Kleene cross (+) operator, and tags.

A rule expansion followed by the asterisk symbol indicates that the expansion may be spoken *zero or more times*. The asterisk symbol is known as the Kleene star (after Stephen Cole Kleene, who originated the use of the symbol). [JSGF] For example,

```
<polite> = please | oh mighty computer;

<command> = <polite>*  don't crash;
```

The rule <command> allows a user to make an utterance like, "Please don't crash," "Oh mighty computer, please, please don't crash," or to ignore politeness with "Don't crash."

A rule expansion followed by the plus symbol indicates the expansion may be spoken *one or more times*. For example,

```
<polite> = please;

<command> = <polite>+  don't crash;
```

The preceding rule requires at least one form of politeness. So, it allows a user to say, "Please, please don't crash." However, "don't crash" is not legal.

Tags provide a mechanism for grammar writers to attach application-specific information to parts of rule definitions. Applications typically use tags to simplify or to enhance the processing of recognition results. Tag attachments do not affect the recognition of a grammar. Instead, the tags are attached to the result object returned by the recognizer to an application. The software interface of the recognizer defines the mechanism for providing tags.

A tag is a unary operator. As such it may be attached to any legal rule expansion. The tag is a string delimited by curly braces '{}'. All characters within the braces are considered a part of the tag, including white space.

The tag attaches to the immediate preceding rule expansion (intervening white space is ignored). For example,

```
<rule> = <action> {tag};
```

As a unary operator, tag attachment has a higher precedence than sequences and alternatives. For example,

```
<publication> = book | magazine | newspaper {thing};
```

The "thing" tag is attached only to the "newspaper" token. Parentheses may be used to modify tag attachment:

```
< publication> = (book | magazine | newspaper) {thing};
```

Unlike the other unary operators, more than one tag may follow a rule expansion. For example,

```
<legalRule> = <action> {tag1} {tag2} {tag3}; // legal
```

Table 4.3 shows two basic grammar examples that define spoken commands, which control a window.

```
grammar com.acme.politeness;
// Body
public <startPolite> = (please | kindly | could you |
oh mighty computer)*;
public <endPolite> = [ please | thanks | thank you ];
```

```
grammar com.acme.commands;
import <com.acme.politeness.startPolite>;
import <com.acme.politeness.endPolite>;
/**
 *  Basic command
 *  @example please move the window
 *  @example open a file
 */
public <basicCommand> = <startPolite> <command>
<endPolite>;
<command> = <action> <object>;
<action> = open | close | delete | move;
<object> = [the | a] (window | file | menu);
```

Table 4.3   Grammar Examples [From JSGF]

E. SUMMARY

This chapter examines the Java Speech API and describes the speech technologies that are supported through Java Speech API. Java Speech Grammar Format (JSGF) and the structure of rule grammars are also examined.

THIS PAGE INTENTIONALLY LEFT BLANK

V. IMPLEMENTATION: BUILDING A NETWORKED, VOICE-ACTIVATED HUMANOID ANIMATION

A. INTRODUCTION

This chapter discusses the initial system structure, development process and implementation details of a networked, voice-activated humanoid animation. An assessment of the final product follows.

The main objective of the initial system is to create humanoid animation driven by human voice. VRML-Java communication makes this objective possible. The development process involves building a motion library, putting the avatars and behaviors together (Interchangeable Actors), voice enabling, networking, and providing available commands and feedback.

B. INITIAL LOW-LEVEL SYSTEM STRUCTURE

System design goals for this project aim to create humanoid animation that a human voice can direct. When a user speaks to a microphone that is connected to the sound card of a computer, the voice input is provided to a speech-recognition application. The speech-recognition application processes this input through predefined acoustic and language models. The speech-recognition engine matches the treated input to the active grammar rules and outputs one or more results or rejects it. Rejection is also considered as a possible result.

The output result of the recognition process is the determinant of the active animation and geometry. The result is compared with the possible conditions, which provides mapping from voice commands to specific predetermined animation behaviors. Nevertheless, the model geometry, which is an avatar in this case, is animated according to the matched condition. Figure 5.1 illustrates initial low-level system structure.

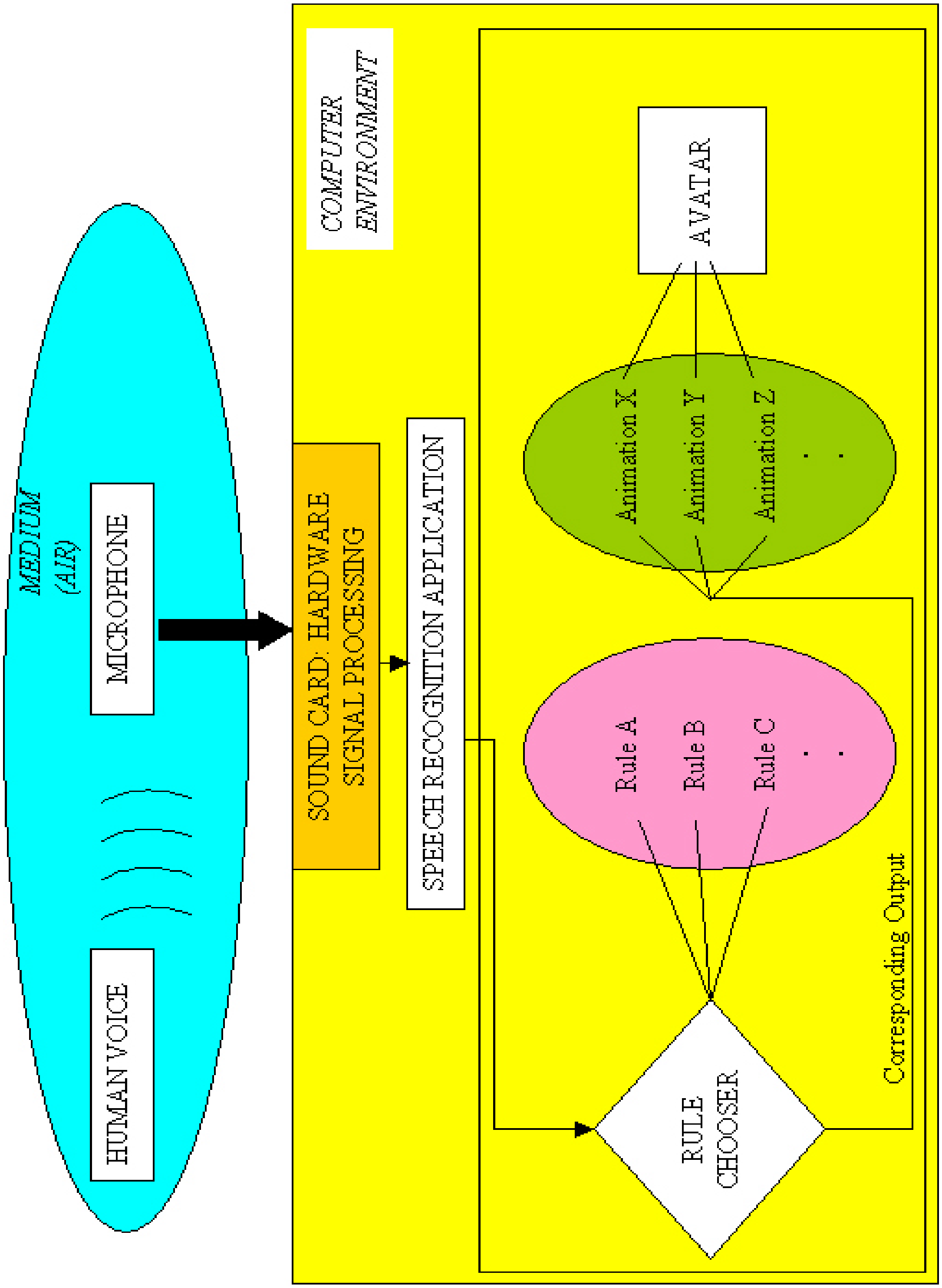


Figure 5.1 Initial Low-Level System Structure of a Voice-Activated Application

C. SYSTEM COMPONENTS

1. IBM VIAVOICE SDK

The ViaVoice SDK is an implementation of the Java™ Speech API. The SDK supports voice-command recognition, dictation, and text-to-speech synthesis, incorporating IBM's ViaVoice speech technology into user interfaces. In much the same way that Java implementations on Windows are built on top of the native Windows GUI capabilities, the ViaVoice SDK, Java Technology Edition is built on top of the native speech recognition and synthesis capabilities in the IBM ViaVoice. The SDK then exposes the standard Java voice interfaces. In order to run the application using the SDK, the user must install an IBM ViaVoice product or ViaVoice executable runtime libraries on the computer. Other implementations of a Voice SDK are possible–this work uses IBM ViaVoice implementation as an excellent current implementation. The IBM ViaVoice SDK is chosen as the basis for building a VUI in this project for two significant reasons:

- The SDK provides access to the ViaVoice engine, which is one of the major speech-recognition engines, and provides the opportunity to build a customized or standardized voice-enabled application.
- The application deriving the SDK classes is implemented in Java, which is a versatile language that enables authors to create and trigger complex animations in VRML/X3D scenes.

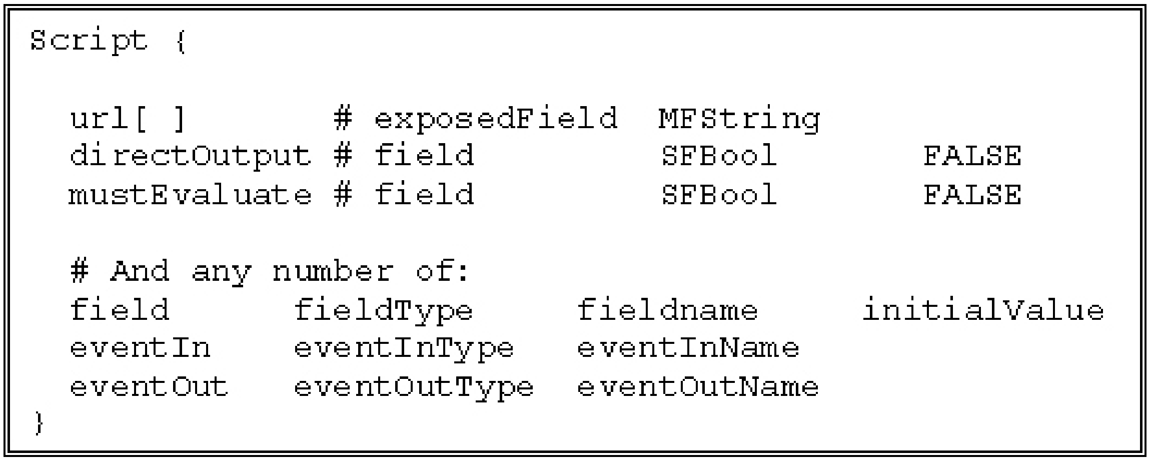
2. Human Models

Since they are compliant with the H-Anim Specifications and they have similar skeleton structures, three avatars, Allen, Nancy and Box Man, formed the human models used in this thesis. These avatars are described in Chapter II.

3. VRML-Java Communication

In the first look, VRML may seem like a limited programming language with predefined nodes suitable only for 3D graphics rendering. Nevertheless, the Script node enables advanced authors to create complex animated 3D scenes. Combining the authoring abilities of VRML with the programming capabilities of Java is possible by integrating code and content via this node.

VRML and Java communicate via Script nodes, which allow authors to connect Java variables to VRML fields. Figure 5.2 shows the basic Script node interface. Type conversion between Java and VRML can be obtained importing `vrml.*` class libraries [DIS-Java-VRML]. However, Java classes, referred by Script nodes, must extend the `vrml.node.Script` class to interface properly with the VRML browser.



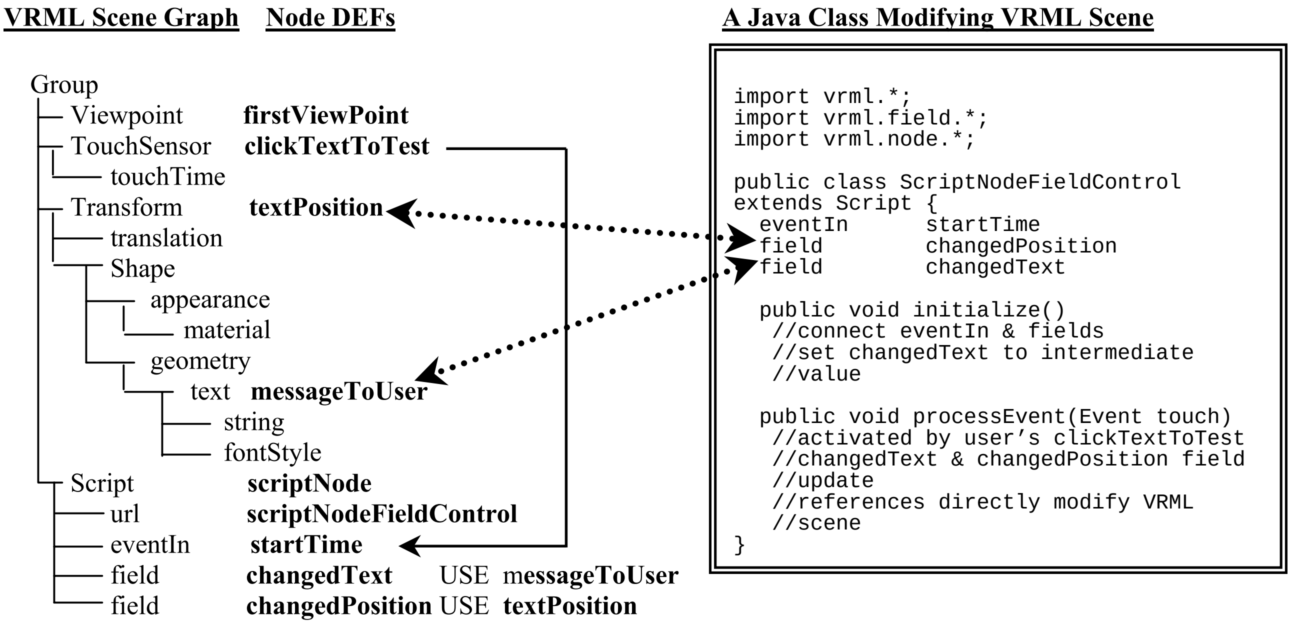
```
Script {

  url[ ]       # exposedField  MFString
  directOutput # field         SFBool       FALSE
  mustEvaluate # field         SFBool       FALSE

  # And any number of:
  field      fieldType     fieldname     initialValue
  eventIn    eventInType   eventInName
  eventOut   eventOutType  eventOutName
}
```

Figure 5.2 Script Node Interface [Ames 97]

The data type `exposedField` indicates that the associated variable has public access, whereas the `field` data type provides only initialization and private access to the respective variable. The `exposedField` data member `url` contains the location of the java class file. This location may be locally on the hard drive or in the Internet. The value of the url exposed field specifies a prioritized list of Uniformed Resource Locators (URLs), ordered from highest priority to lowest. If the browser cannot find the named file in the first location, it tries the second URL in the list, and so on. The fields `directOutput` and `mustEvaluate` are hints to the browser on how to optimize performance. When `directOutput` is set to FALSE, the script may read, but not write the value of any `exposedField` and `eventOut` for any node object to which it has access. Conversely, when the value of the `directOutput` field is TRUE, the program script may also write the value of any exposedField and send a value to any eventIn of any node value to which it has access. If mustEvaluate is FALSE, the browser may postpone program script evaluation. Setting this value to TRUE, forces the browser to update when a new value is received by an eventIn for the node. The data types eventIn and eventOut are events, which provide interactivity and fluidity to the VRML scenes. Events are time-stamped values of data types, and eventIn data types of a target node must match exactly the ROUTEd eventOut data types from a source node. When a pre-defined event is triggered, the value of the variable is sent along with a time-stamp from the eventOut connection to the associated eventIn connection. Figure 5.3 shows an example of VRML-Java communication.



```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class ScriptNodeFieldControl
extends Script {
  eventIn      startTime
  field        changedPosition
  field        changedText

  public void initialize()
   //connect eventIn & fields
   //set changedText to intermediate
   //value

  public void processEvent(Event touch)
   //activated by user's clickTextToTest
   //changedText & changedPosition field
   //update
   //references directly modify VRML
   //scene
}
```

Figure 5.3 An Example of VRML-Java Communication [Brutzman 98]

Upon starting the VRML scene, the associated java class identified by the script node's url field is accessed, and its public method initialize() is called automatically. In this method, the fields passed by reference from the VRML file are connected to the eventIn and field variables. The programmer may also perform any initialization that is deemed necessary, such as positioning or content changes. When the user activates the TouchSensor named clickTextToTest by clicking on the text with the mouse, an event and time-stamp is sent from touchTime's `eventOut` to the script node's `eventIn` startTime. This step calls the script node's public method `processEvent()`, which can then perform any desired computations. The programmer can then perform any java functionality that is desired and modify the fields passed in by reference accordingly. In this example, both the content and position of the text string is modified. [Dutton 2001]

D. DEVELOPMENT PROCESS

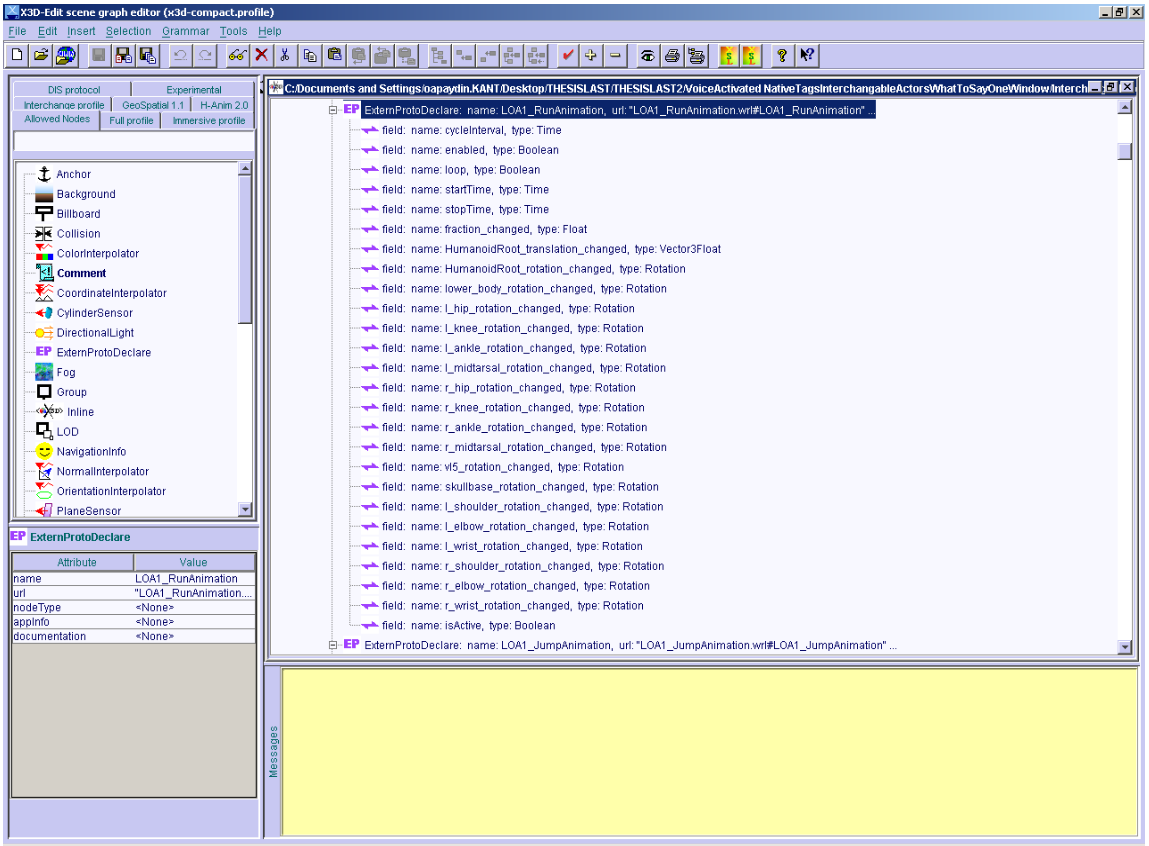
1. Building a Motion Library

A motion library is needed to animate the avatars; Allen, Nancy and Box Man. This motion library will be the basis for interchangeable animations. Nancy's animation behaviors are implemented as independent five behaviors; *Stand*, *Walk*, *Run*, *Jump* and *Kneel*. These behaviors are created using Prototype definitions and intended to be reusable by other humanoids. The created behaviors are examples of forward kinematics, that is, each joint angle corresponding to that behavior was specified by the author.

Prototype nodes in VRML enables authors to create new node types, which consist of a *node interface* and a *node body*. Fields, exposedFields, eventIns, and eventOuts, similar to the program script interface declared in a Script node, describe the node interface. The node body describes what the node does and how it does it. A node body is defined using any combination of predefined VRML and other prototype nodes. Prototype recursion is not allowed.

A *ProtoDeclare* defines a new node type that can be used anywhere in the rest of the same file. There are times, however, when it is more convenient to put ProtoDeclare in an external file, such as when maintaining a library of new node types. In these cases, this ProtoDeclare can be accessed by using an *ExternProtoDeclare* in a different file. This new node can then be used and instantiated by *ProtoInstance* in the rest of the file.

ProtoDeclares of the animation behaviors are placed in separate files and in this way, a motion library is constructed. This library is online at http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimation/chapt er.html. To use one of these animation prototypes in a different humanoid .x3d file, the node definition can be referenced by using ExternProtoDeclare (external prototype). ExternProtoDeclare includes a URL or a list of URLs that references an external file containing the corresponding ProtoDeclare. When the VRML browser encounters the ExternProtoDeclare, it finds the new node-type definition in the file specified by the URL. That new node type is then available for use anywhere in the rest of the file. Figure 5.4 shows an example of ExternProtoDeclare, which references one of the behavior Protos.



EP ExternProtoDeclare

Attribute	Value
name	LOA1_RunAnimation
url	"LOA1_RunAnimation....
nodeType	<None>
appInfo	<None>
documentation	<None>

```
EP ExternProtoDeclare:  name: LOA1_RunAnimation,  url: "LOA1_RunAnimation.wrl#LOA1_RunAnimation" ...
   field:  name: cycleInterval,  type: Time
   field:  name: enabled,  type: Boolean
   field:  name: loop,  type: Boolean
   field:  name: startTime,  type: Time
   field:  name: stopTime,  type: Time
   field:  name: fraction_changed,  type: Float
   field:  name: HumanoidRoot_translation_changed,  type: Vector3Float
   field:  name: HumanoidRoot_rotation_changed,  type: Rotation
   field:  name: lower_body_rotation_changed,  type: Rotation
   field:  name: l_hip_rotation_changed,  type: Rotation
   field:  name: l_knee_rotation_changed,  type: Rotation
   field:  name: l_ankle_rotation_changed,  type: Rotation
   field:  name: l_midtarsal_rotation_changed,  type: Rotation
   field:  name: r_hip_rotation_changed,  type: Rotation
   field:  name: r_knee_rotation_changed,  type: Rotation
   field:  name: r_ankle_rotation_changed,  type: Rotation
   field:  name: r_midtarsal_rotation_changed,  type: Rotation
   field:  name: vl5_rotation_changed,  type: Rotation
   field:  name: skullbase_rotation_changed,  type: Rotation
   field:  name: l_shoulder_rotation_changed,  type: Rotation
   field:  name: l_elbow_rotation_changed,  type: Rotation
   field:  name: l_wrist_rotation_changed,  type: Rotation
   field:  name: r_shoulder_rotation_changed,  type: Rotation
   field:  name: r_elbow_rotation_changed,  type: Rotation
   field:  name: r_wrist_rotation_changed,  type: Rotation
   field:  name: isActive,  type: Boolean
EP ExternProtoDeclare:  name: LOA1_JumpAnimation,  url: "LOA1_JumpAnimation.wrl#LOA1_JumpAnimation" ...
```

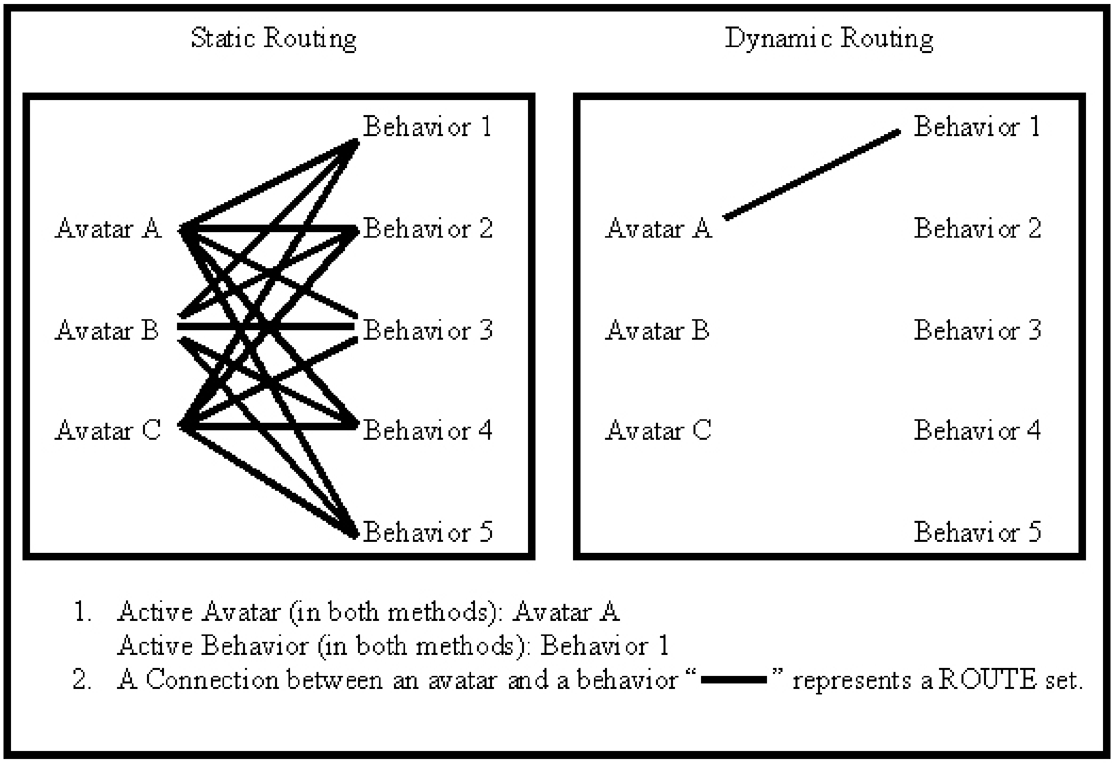
Figure 5.4 X3D-Edit Screen Snapshot of *Run* ExternProto http://www.web3d.org/TaskGroups/x3d/translation/examples/HumanoidAnimation/LOA1_RunAnimation.x3d

2. Putting the Avatars and Behaviors Together: The Interchangeable Actors

One of the most important goals of this thesis is to create an example that demonstrates interchangeable humanoids and animations. This example was built by employing three avatars (Allen, Nancy, and Box Man) and five animation behaviors (*Stand*, *Walk*, *Run*, *Jump*, and *Kneel*).

Since a visual editor provides a significant advantage for the author to reduce the errors when dealing with complex structures like humanoids, X3D-Edit was used to implement this project. As the first step, Allen was converted to X3D native tags because an X3D version of it did not previously exist. Then the avatars were put in the same scene in such a way that the active one could switch to any of the others. The motion library, mentioned previously, is referenced by using ExternProtoDeclares. The behavior prototypes were then instantiated. Texts, to which touch sensors are attached, are also added to the scene to activate an avatar or a behavior.

*ROUTE*s are wires, which connect eventIns to eventOuts and carry events from one to another. Hard wiring (static routing) was one of the available options in implementing. However this method had some drawbacks such as performance degradation (continuous event exchange between inactive avatars and active behavior) and expanding complexity (hard to add new avatars and behaviors). Instead, *Dynamic Routing* was used to eliminate these disadvantages. A script (JavaScript) was written to provide the connections between the avatars and the behaviors. Only active behavior and avatar were wired within the script. First, avatar, avatar joint, and animation behavior DEFs were indexed. Then, relevant connections between avatar joints and animation behavior interpolators were created and deleted using the `addRoute()` and `deleteRoute()` methods of `Browser` class (refer to [VRML 97]) to animate the active avatar with the active animation behavior. The ROUTE redundancy in Static Routing can be clearly seen in Figure 5.5.



1. Active Avatar (in both methods): Avatar A
   Active Behavior (in both methods): Behavior 1
2. A Connection between an avatar and a behavior "▬▬" represents a ROUTE set.

Figure 5.5 Static Routing vs. Dynamic Routing

When the user clicks on the relevant text to change the current avatar or behavior, the script gets the event and accordingly creates a new ROUTE set between the active avatar and behavior after deleting the old Routes. Besides dynamic routing operations, the script also handles avatar switching.

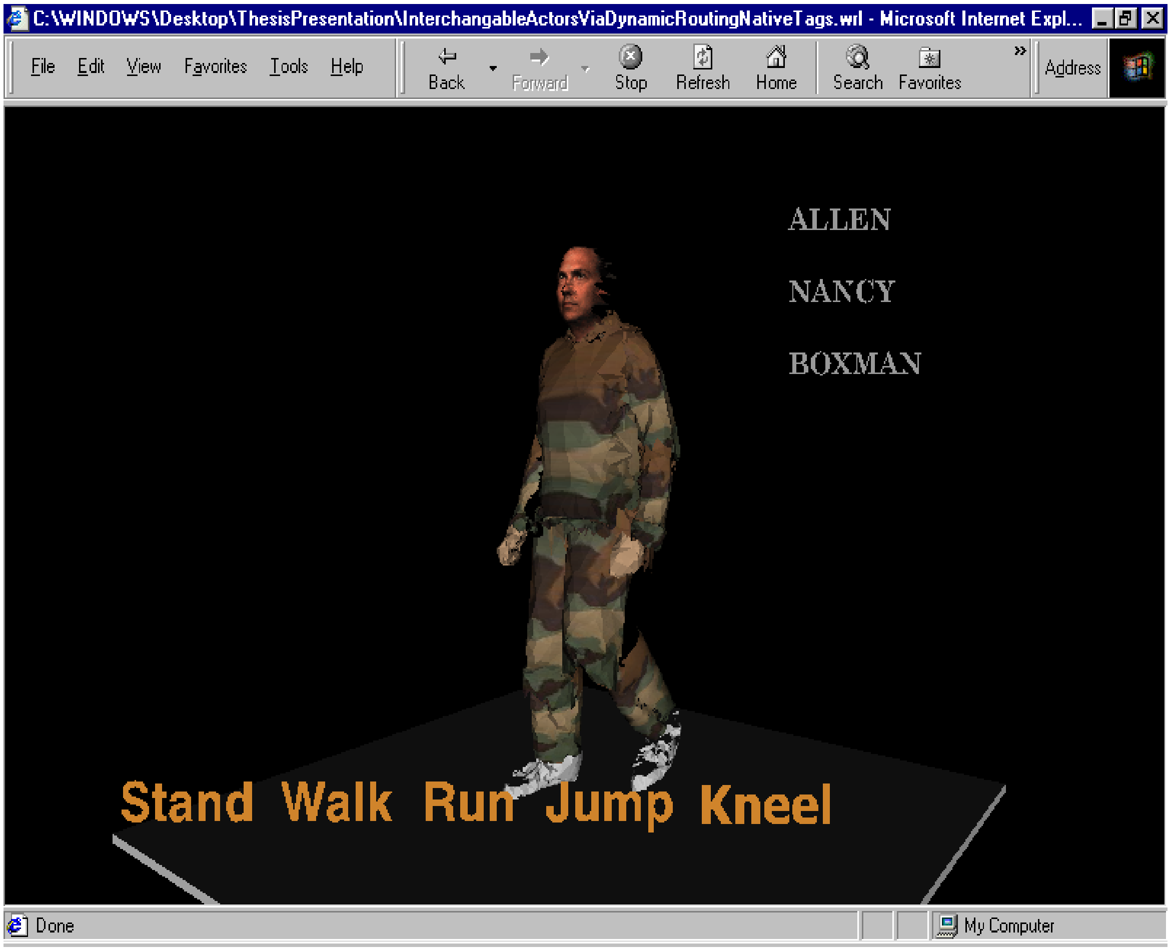


Figure 5.6 Screen Shot of Interchangeable Actors

3. Voice Enabling

The Nancy avatar was the first example to be connected to voice activation during this project. Instead of using the mouse for selections, a user can state one of the available commands to animate Nancy. These one-word commands consisted of four animation behaviors: *Stand*, *Walk*, *Run* and *Jump*. Later, the Allen avatar was voice-activated in the same way and a *Kneel* behavior was further added. These two successful experiences led to a more complex task: Voice Activation of Interchangeable Actors.

IBM ViaVoice SDK, an implementation of Java Speech API, was used to build a suitable VUI. A speech recognizer class in Java was written to create necessary objects and allocate required resources for speech recognition and synthesizing. Moreover, the limit of the dialogue between user and the application must be defined. Since this is an exemplar command and control application, commands available to the user were determined. These commands, consisting of one or more words, were divided into two groups. The first group was used to initiate the application (opening the browser and loading the scene) and the second group was intended for animation (activating avatars and behaviors). Based on the determined commands, a grammar in Java Speech Grammar Format (refer to Chapter IV) was built for this application. The grammar rules were designed so that they provided more than one command option for a specific task. Table 5.1 indicates legal voice commands in this application. The grammar was located on a text file separate from the source code. Otherwise, any change of one of the grammar rules required recompiling the code. The grammar rules are read from this text file, and then parsed and enabled in the recognizer class. The speech-recognition application takes the user's voice command, tries to match the input to one of the active rules and provides an array of results and their tags. Then, the best result's tag is sent to a client application through the network. The reason for sending tags instead of results is that a tag may represent multiple results, which flag the same task and consequently makes handling the incoming packet content on the client side easier.

The recognition application also has a synthesizer object that greets the user when the application is initiated. Additionally, this object provides feedback to the user when the recognition engine cannot match the voice input to one of the active grammar rules, that is, when the result is rejected.

Welcome Phase Commands	**Animation Phase Commands**
	Nancy
Initiate Actors	Allen
	Box Man
Initiate Humanoid Animation	Switch To Nancy
	Switch To Allen
Initiate Avatars	Switch To Box Man
	Display Nancy
Open Actors	Display Allen
	Display Box Man
Open Humanoid Animation	Turn Into Nancy
	Turn Into Allen
Open Avatars	Turn Into Box Man
	Jump
Start Actors	Run
	Walk
Start Humanoid Animation	Kneel
	Stand
Start Avatars	Why Don't You Jump
	Why Don't You Run
	Why Don't You Walk
	Why Don't You Kneel
	Why Don't You Stand
	Could You Jump
	Could You Run
	Could You Walk
	Could You Kneel
	Could You Stand
	Please Jump
	Please Run
	Please Walk
	Please Kneel
	Please Stand
	Good bye
	So long

Table 5.1 Recognized Voice Commands during Welcome Phase and Animation Phase

4. Networking

Creating a networked application is also one of the primary goals of this thesis. The DIS-Java-VRML package implements and simplifies VRML-Java communication, as previously discussed in this chapter.

User Datagram Protocol / Internet Protocol (UDP/IP) is used as the network protocol. UDP provides a connectionless transmission and best-effort delivery. With UDP/ IP, an application can direct a packet to be sent to one other application endpoint. Although UDP/IP is not as reliable as Transmission Control Protocol (TCP), it removes most of the overhead introduced by TCP [Zyda 99].

The designed networking system consists of one server and two client applications. The server application (SpeechRecognizer) sends DatagramPackets to the network. In this implementation, a DatagramPacket includes a destination and a matched rule tag, which is the result of the recognition process. With the first client application (InvocationClient), to open the browser, to load the VRML scene and to quit the client side applications are aimed. The server application sends the rule tags, related with the initiation of the scene, to the first client. Then, this client application invokes the browser by creating it as a Process object. Besides, the invoked browser loads the VRML scene. Once the browser loads the VRML file, the OrderExecutor class is executed through the Script node. The OrderExecutor class starts the second client (ClientNetListener) as a thread. This time, the server sends DatagramPackets to the second client. The second client passes the incoming packet content to the OrderExecutor class. According to the packet content, the OrderExecutor class affects the relevant VRML nodes for demanded animation (refer to VRML-Java communication). Figure 5.7 demonstrates a flowchart of this networking process.

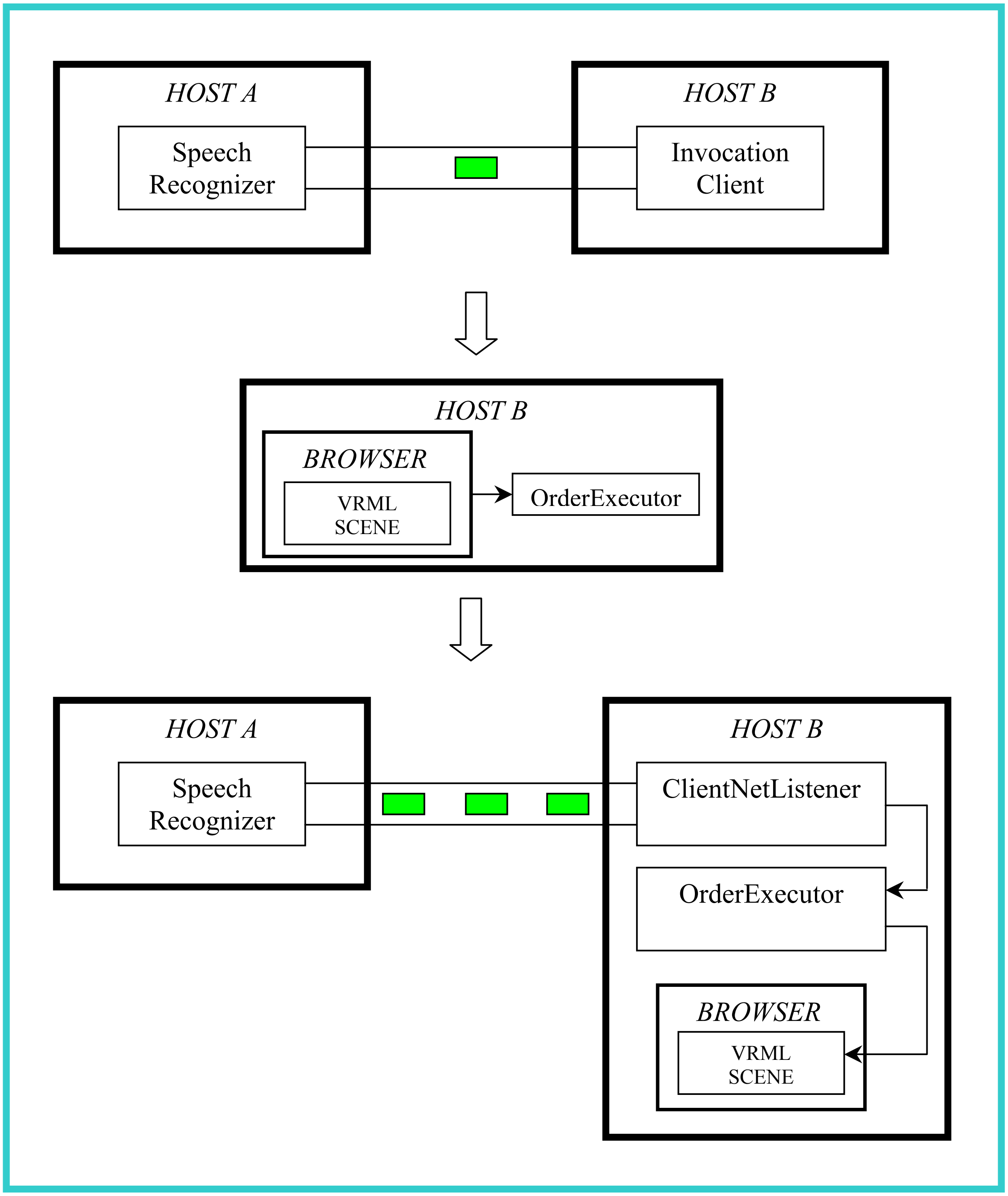


Figure 5.7 Flowchart of Networked Humanoid Animation

A user may open the browser, which displays the VRML scene and may also animate the available avatars in a remote host with his or her voice commands. Then, he or she may quit the application without clicking or typing. (See Figure 5.8)

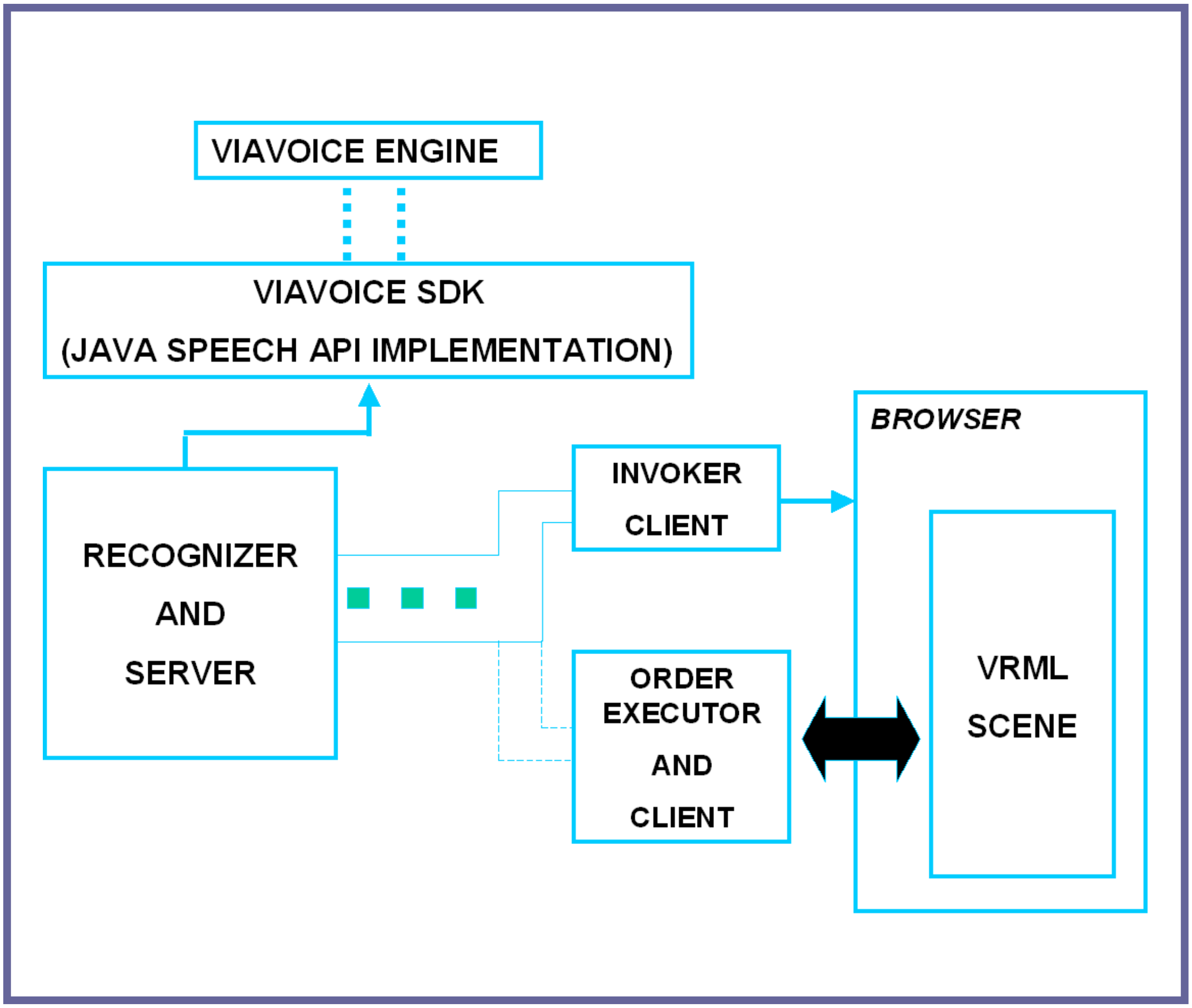
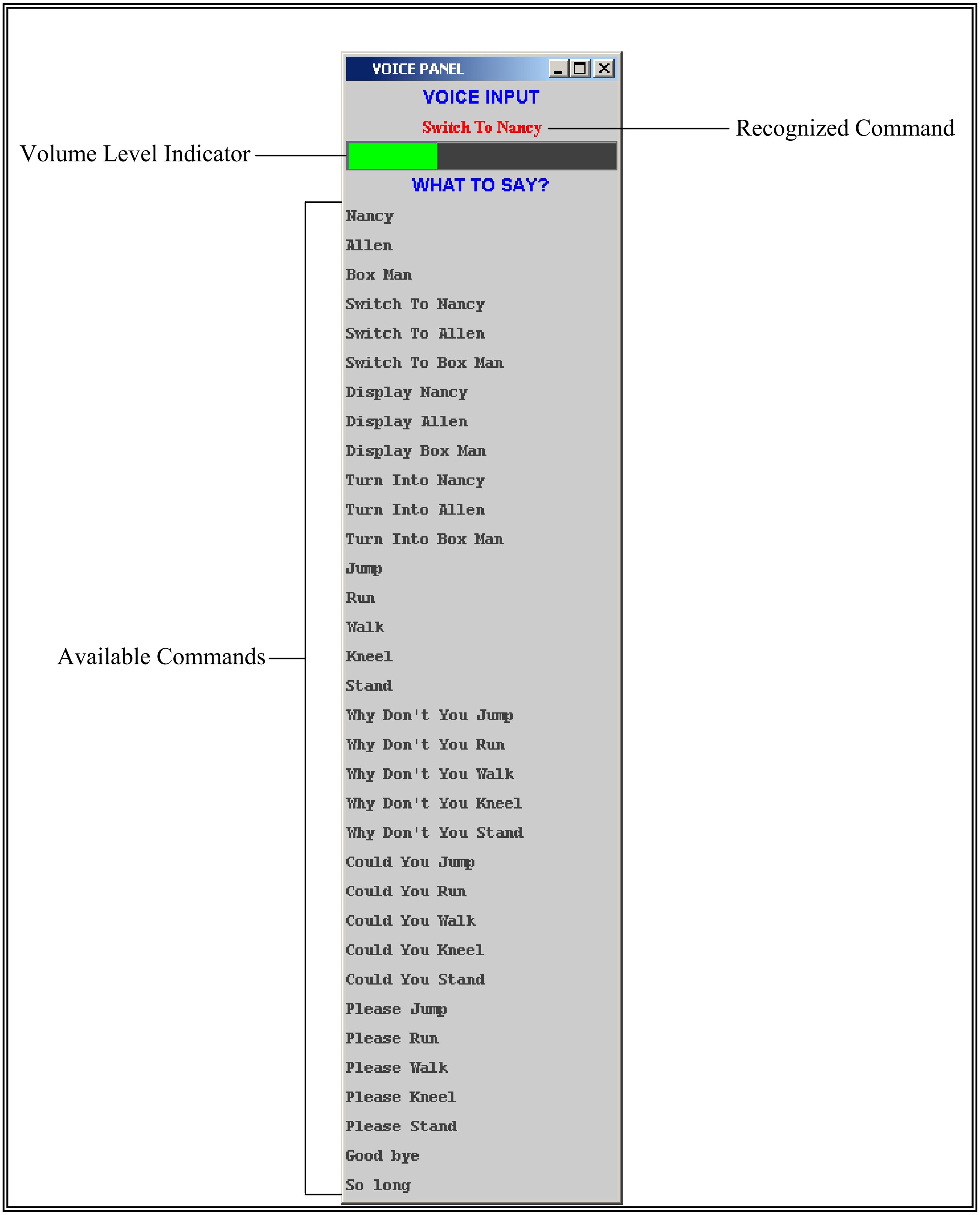


Figure 5.8 Networked, Voice-Enabled Humanoid Animation System Structure

5. Providing Available Commands and Feedback

The user should know what to say to the application. If the number of the available commands is excessive, displaying them becomes a must. Hence, some GUI components showing the available commands are employed. A message that indicates what is said (recognized command) and a volume-level indicator are also added to these GUI components for feedback purposes. The Voice Panel automatically adjusts its size according to the screen resolution once it is initiated. Figure 5.9 demonstrates Voice Panel and Figure 5.10 shows a screen shot of the Voice Activated Interchangeable Actors.



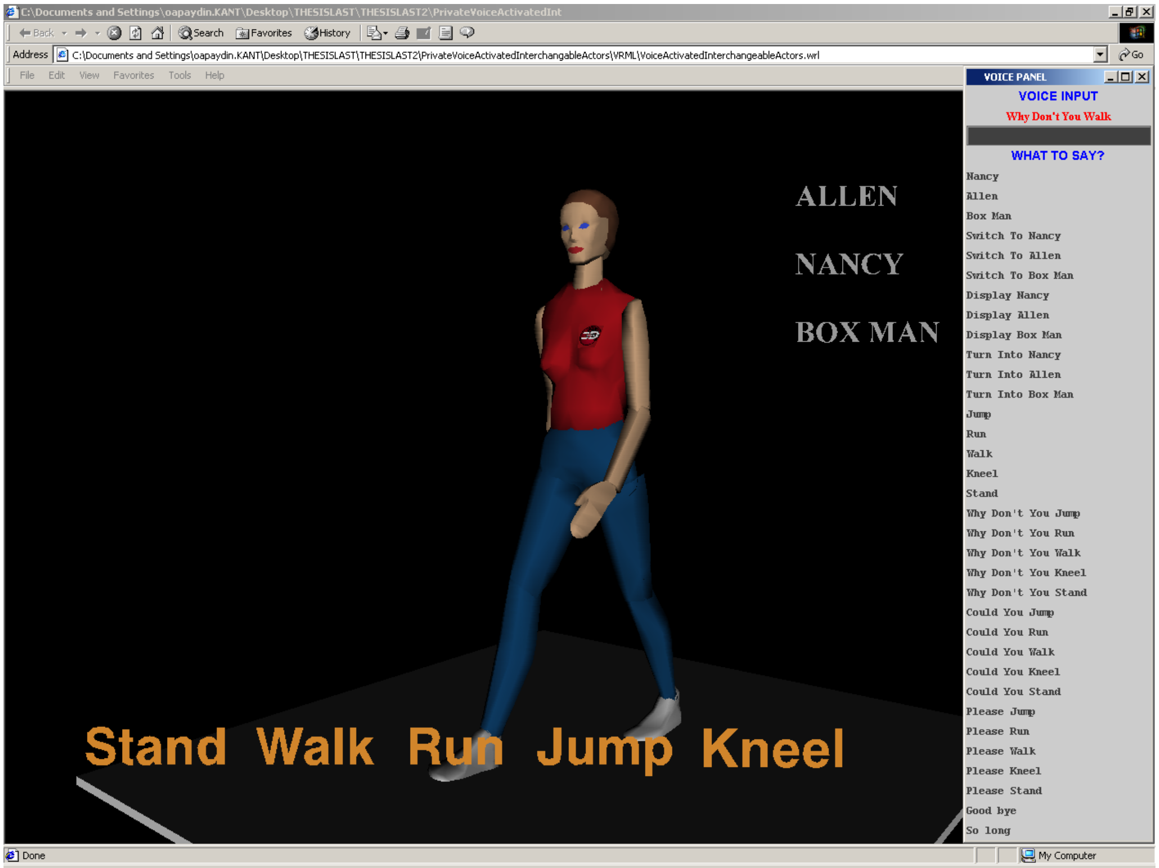
VOICE INPUT

Switch To Nancy

WHAT TO SAY?

Nancy
Allen
Box Man
Switch To Nancy
Switch To Allen
Switch To Box Man
Display Nancy
Display Allen
Display Box Man
Turn Into Nancy
Turn Into Allen
Turn Into Box Man
Jump
Run
Walk
Kneel
Stand
Why Don't You Jump
Why Don't You Run
Why Don't You Walk
Why Don't You Kneel
Why Don't You Stand
Could You Jump
Could You Run
Could You Walk
Could You Kneel
Could You Stand
Please Jump
Please Run
Please Walk
Please Kneel
Please Stand
Good bye
So long

Figure 5.9 Voice Panel Showing Animation Phase Commands



Address: C:\Documents and Settings\oapaydin.KANT\Desktop\THESISLAST\THESISLAST2\PrivateVoiceActivatedInterchangableActors\VRML\VoiceActivatedInterchangeableActors.wrl

VOICE INPUT: Why Don't You Walk

WHAT TO SAY?

```
Nancy
Allen
Box Man
Switch To Nancy
Switch To Allen
Switch To Box Man
Display Nancy
Display Allen
Display Box Man
Turn Into Nancy
Turn Into Allen
Turn Into Box Man
Jump
Run
Walk
Kneel
Stand
Why Don't You Jump
Why Don't You Run
Why Don't You Walk
Why Don't You Kneel
Why Don't You Stand
Could You Jump
Could You Run
Could You Walk
Could You Kneel
Could You Stand
Please Jump
Please Run
Please Walk
Please Kneel
Please Stand
Good bye
So long
```

Figure 5.10 Screen Shot of Voice-Activated Interchangeable Actors (Nancy and Walk selected)

E. ASSESSMENT OF THE FINAL PRODUCT

The features of the final product can be summarized as

- Hybrid: It involves both VUI and GUI components.
- Networked (UDP/IP): A user may command and control the client applications in a remote host.
- User-Independent: The speech recognizer application does not require any training to understand the user’s speech patterns.
- Mono-Lingual: It understands only English commands.
- Multi-Platform: Since only Java and VRML are used in implementing, the final product may run on various platforms and operating systems.

Different users experienced the final product. Two of the experiences will only be discussed because they are extreme examples of the varied performances. The first one took place in front of an audience in the final academic presentation of this thesis. The audience was well informed about the factors affecting the accuracy (specifically noise) by the presenter. A person from the audience was called for the demo. Before the presentation, the participant was given some important tips, such as "no shouting, speak naturally, no need to speak discretely," etc. The participant followed the tips and 100% accuracy was observed. In another experience, a participant, who had no background with speech-recognition applications and had an atypical voice, tested the application. The participant was speaking discretely and loud. Moreover, the audience in the background was talking continuously. As a result, 20% accuracy, which is very low, was observed. These two examples present how important the user's training is, and how noise affects the accuracy.

IBM ViaVoice SDK supports French, German, Italian, Spanish, UK English, and US English. Therefore, the final product is promising for limited internationalization (I18N). However, the current implementation, grammar, and user interfaces must be modified when integrating this capability. I18N testing is a promising area for future work.

F. SUMMARY

This chapter discussed the design issues, implementation details, development process and provided an assessment of the final product.

VI. CONCLUSION AND FUTURE WORK

A. THESIS CONCLUSIONS

1. Integration of Speech-Recognition Technology to the Networked Virtual Environments (Net-VEs)

As computation power increases, integrating the speech-recognition technology to the Networked Virtual Environments (Net-VEs) is possible. This thesis demonstrates a way of realizing this objective. Improvements in speech-recognition technology to build more robust speech-recognition applications will expedite this integration.

2. Hybrid Interfaces (VUI + GUI)

The applications incorporating VUIs and GUIs in an appropriate way can be very powerful. While a VUI provided access to the application, the GUI components obtained feedback and displayed available commands for the user in this thesis. Hybrid interfaces may also augment the speed to access an application. For example, when using an application in a hybrid interfaced operating system, a user might need to open another application. In this case, the user orders the operating system to do it with his or her voice. This capability eliminates the overhead introduced by the GUI-dependent operating systems, such as minimizing the window, finding the application shortcut and starting it, etc.

3. Interchangeable Humanoids and Animation Behaviors

Humanoids and animations can be authored independently. Choosing humanoids, which have identical skeleton structures, reduces the complexity in creating interchangeable humanoids and animation behaviors. Reusable animation behaviors can be created, prototyped and applied to various humanoids, which are built according to standard specifications. When dealing with interchangeability of the humanoids and behaviors, the efficiency of the implementation becomes very important. Building archived libraries of humans and behaviors will become increasingly valuable.

B. RECOMMENDATIONS FOR FUTURE WORK

1. Building Simulation of a Scenario or a Game

The next step in this research is to build a simulation or a game, which takes advantage of VUIs. Besides accessing the application with voice, avatars that can fulfill voice commands may also be inserted to the simulation or the game. Then the available entities in the simulation or game can be directed by the voice of the user. The programmer must carefully design commands available to the user by considering the usability of the application.

2. Improving Networking

According to the chosen scenario or game, present networking implementation may be changed and improved. UDP/IP, used as the network protocol in this project, offers no reliability or ordering guarantees. In the scope of a simulation or a game to be implemented, a more reliable protocol, which offers a customized packet, can be used.

3. Expanding the Motion Library

This thesis included building a reusable motion library. This library consists of limited number of behaviors. Therefore, the motion library can be easily expanded with motion tracking systems. Using motion-tracking systems to obtain new animation data also contributes to the realism of the behavior.

4. Composition of Animation Behaviors

The user might want to combine two available animation behaviors. For example, he or she might say "Run and Jump." Current application does not handle this situation. This issue can be addressed in the following ways by:

- Designing a hierarchical structure for animation behaviors so that the behavior with a higher precedence is preferred on top of the behavior, with a lower precedence.
- Creating a hybrid behavior that includes the characteristics of both behaviors.

5. Agents

Agents are like software nano-robots. They give designers the ability to solve problems indirectly, from inside the virtual representation of the problem. They also give the problem solver a low-level, detail rich view of the problem or model. Agents adapt, learn and take action based on local information at the lowest level of the problem or model [Hiles 2001]. Agents and speech-recognition technologies can be used together to emulate human behavior and to provide intelligent virtual characters in Virtual Environments (VEs).

6. Support for Tactical Applications

Speech-recognition technology can improve team communication in diverse and hazardous environments. Noise-cancellation headsets can lead to protection of operators in life-threatening situations. Augmented reality (AR) can provide a great deal of operational support information to team members when aural or visual perception is weak or impossible. Moreover, body-tracking systems can send the position, orientation and behavior information of team members in a hazardous area to a central visualization system to be reviewed by the experts in real time. Visual display of this information bundle can provide a great deal of support in making critical decisions and save lives.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – ACRONYMS

API	Application Programming Interface
AR	Augmented Reality
DAG	Directed Acyclic Graph
DARPA	Defense Advanced Research Project Agency
DOD	Department of Defence
DTD	Document Type Definition
EXTERNPROTOs	External Prototypes
FBCB2	Force XXI Battle Command, Brigade and Below
GUIs	Graphical User Interfaces
H-Anim	Humanoid Animation
IP	Internet Protocol
ISO	International Standards Organization
I18N	Internationalization
JSAPI	Java Speech API
JSGF	Java Speech Grammar Format
Net-VEs	Networked Virtual Environments
NLP	Natural Language Processing
NPS	Naval Postgraduate School
PDAs	Personal Digital Assistants
PROTOs	Prototypes
SALT	Size, Activity, Location, Time
SUR	Speech Understanding Research
TCP	Transmission Control Protocol

TTS	Text-to-Speech
UDP	User Datagram Protocol
URLs	Uniformed Resource Locators
VEs	Virtual Environments
VRML	Virtual Reality Modeling Language
VUIs	Voice User Interfaces
X3D	Extensible 3D

APPENDIX B – 3D SCENES AND ANIMATION BEHAVIORS

All 3D scenes and animation behaviors are included in the Source Code CD. The source code files and directories in the Source Code CD are

- ClientNetListener.java, .class
- Invocation Client.java, .class
- OrderExecutor.java, .class
- Speech Recognizer.java, .class
- VoicePanel.java, .class
- Rule.txt
- Readme.html
- Under X3D directory;

  VoiceActivatedInterchangeableActors.x3d

  LOA1_JumpAnimation.x3d

  LOA1_KneelAnimation.x3d

  LOA1_RunAnimation.x3d

  LOA1_StandAnimation.x3d

  LOA1_WalkAnimation.x3d

- Under VRML directory

  VoiceActivatedInterchangeableActors.wrl

  LOA1_JumpAnimation.wrl

  LOA1_KneelAnimation.wrl

  LOA1_RunAnimation.wrl

  LOA1_StandAnimation.wrl

  LOA1_WalkAnimation.wrl

- slides directory
- centers directory
- javadoc directory
- lib directory

APPENDIX C – CD DISTRIBUTION LIST

1. Michael Zyda
   Chair, MOVES Academic Group
   Naval Postgraduate School
   Monterey, California

2. Rudy Darken
   Naval Postgraduate School
   Monterey, California

3. Don Brutzman
   Naval Postgraduate School
   Monterey, California

4. Xiaoping Yun
   Naval Postgraduate School
   Monterey, California

5. Chris Darken
   Naval Postgraduate School
   Monterey, California

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[Ames 97] Ames, A.L., Nadeu, D.R., and Moreland, J.L., "VRML2.0 Sourcebook", John Wiley& Sons, Inc, 1997.

[Brutzman, Blais, Horner, Nicklaus 2001] Brutzman D. P., Blais C., Horner D., Nicklaus S., "Web-Based 3D Technology for Scenario Authoring and Visualization: The SAVAGE Project", 2001.

[DIS-Java-VRML] DIS-Java-VRML Working Group, http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/

[Dutton 2001] Dutton, A., "Developing Articulated Human Models from Laser Scan Data for Use as Avatars in Real-Time Networked Virtual Environments", M.S. Thesis, 2001.

[Fulton 2000] Fulton, S. M., "Speak Softly, Carry a Big Chip", The New York Times, March 30, 2000.

[H-Anim 1.1] H-Anim 1.1 Specification, http://www.h-anim.org/Specifications/H-Anim1.1/

[H-Anim 2001] H-Anim 2001 Specification, http://www.h-anim.org/Specifications/H-Anim2001/

[Hiles 2001] Hiles, J., "Software Agents: Smarter, Easier to Create, More Capable", 2001, http://www.movesinstitute.org/OpenHouse2001/Presentations/Hiles.ppt

[JSAPI 1.0] Java™ Speech API 1.0 Specification and Java Speech API Programmer's Guide, http://java.sun.com/products/java-media/speech/

[JSGF] Java Speech Grammar Format Specification, http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html

[Kavanagh 95] Kavanagh, B., "Speech Recognition", 1995

[Kemble 2001] Kemble, K., A., "An Introduction to Speech Recognition", October 2001

[Long 94] Long, B., "Natural Language as an Interface Style", University of Toronto, May 1994.

[Maurer] Maurer, J., "History of Speech Recognition", Stanford University.

[Miller 2000] Miller, T., E., "Integrating Realistic Human Group Behaviors Into a Networked 3D Virtual Environment", September 2000.

[NetByTel] NetByTel, "History of Speech Recognition", http://www.netbytel.com/literature/e-gram/technical3.htm

[Nielsen 99] Dr. Nielsen, J., "Will Voice Interfaces Replace Screens",1999.

[PCMagazine 99] PC Magazine UK, "Speech Recognition Software", July 1999.

[Pocock, Rosebush 2002] Pocock, L., Rosebush J., "The Computer Animator's Technical Handbook", 2002.

[Seffers 2001] Seffers, G. I., "The Voice of Combat", Technology Trends August 6, 2001.

[Simpson 99] Dr. Simpson, Charles B., "What Makes Each Human Voice Distinct?", 1999.

[VRML 97] The Virtual Reality Modeling Language Specification, 1997, http://www.web3d.org/Specifications/VRML97/

[Zyda 99] Zyda, M., Singhal S., "Networked Virtual Environments, Design and Implementation", 1999.

INITIAL DISTRIBUTION LIST

1. Deniz Kuvvetleri Komutanligi
   Personel Egitim Daire Baskanligi
   Bakanliklar, Ankara
   TURKEY

2. Defense Technical Information Center
   Ft. Belvoir, Virginia

3. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

4. Deniz Harp Okulu Komutanligi Kutuphanesi
   Tuzla, Istanbul
   TURKEY

5. Michael Zyda
   Chair, MOVES Academic Group
   Naval Postgraduate School
   Monterey, California

6. John Hiles
   Naval Postgraduate School
   Monterey, California

7. Rudy Darken
   Naval Postgraduate School
   Monterey, California

8. Don Brutzman
   Naval Postgraduate School
   Monterey, California

9. Xiaoping Yun
   Naval Postgraduate School
   Monterey, California

10. John Falby
    Naval Postgraduate School
    Monterey, California

11. Chris Darken
Naval Postgraduate School
Monterey, California

12. Lynn Pocock
77 Fornelius Ave.
Clifton, New Jersey 07013